COMPUTATIONAL ARCHITECTURES INTEGRATING NEURAL AND SYMBOLIC PROCESSES

A Perspective on the State of the Art

EDITED BY
RON SUN
LAWRENCE A. BOOKMAN

FOREWORD BY
MICHAEL A. ARBIB

KLUWER ACADEMIC PUBLISHERS

COMPUTATIONAL ARCHITECTURES INTEGRATING NEURAL AND SYMBOLIC PROCESSES

A PERSPECTIVE ON THE STATE OF THE ART

THE KLUWER INTERNATIONAL SERIES
IN ENGINEERING AND COMPUTER SCIENCE

COMPUTATIONAL ARCHITECTURES INTEGRATING NEURAL AND SYMBOLIC PROCESSES

A PERSPECTIVE ON THE STATE OF THE ART

EDITED BY

Ron Sun
The University of Alabama
Tuscaloosa, AL, USA

■

Lawrence A. Bookman
Sun Microsystems Laboratories
Chelmsford, MA, USA

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Contents

Contributors

John A. Barnden
Computing Research Laboratory
and Computer Science Department
New Mexico State University
Las Cruces, New Mexico 88003
jbarnden@nmsu.edu

Lawrence A. Bookman
Sun Microsystems Laboratories
Chelmsford, MA 01824
lbookman@east.com.sun

Garrison W. Cottrell
Department of Computer Science
and Engineering
Institute for Neural Computation
University of California, San Diego
La Jolla, CA 92093-0114
gary@cs.ucsd.edu

Michael G. Dyer
Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, CA 90024
dyer@cs.ucla.edu

James Hendler
Department of Computer Science
University of Maryland
College Park, Maryland 20742
hendler@cs.umd.edu

Vasant Honavar
University of Iowa
Iowa State University
Ames, Iowa 50011
honavar@cs.iastate.edu

Stuart A. Jackson
Computer Science Department
Regent Court
University of Sheffield
S1 4DP, Sheffield, UK
S.Jackson@dcs.shef.ac.uk

Chris Lacher
Department of Computer Science
Florida State University
Tallahassee, FL 32306
lacher@cs.fsu.edu

Trent Lange
Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, CA 90024
lange@cs.ucla.edu

Charles Lin
Department of Computer Science
University of Maryland
College Park, Maryland 20742
clin@cs.umd.edu

Risto Miikkulainen
Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
risto@cs.utexas.edu

David C. Noelle
Department of Computer Science
and Engineering
Institute for Neural Computation
University of California, San Diego
La Jolla, CA 92093-0114
dnoelle@cs.ucsd.edu

K.D. Nguyen
Department of Computer Science
Florida State University
Tallahassee, FL 32306
nguyen@cs.fsu.edu

Noel E. Sharkey
Computer Science Department
Regent Court
University of Sheffield
S1 4DP, Sheffield, UK
N.Sharkey@dcs.shef.ac.uk

Ron Sun
Department of Computer Science
College of Engineering
The University of Alabama
Tuscaloosa, AL 35487
rsun@cs.ua.edu

Foreword

The "Integration of Neural and Symbolic Processes" has been with us ever since the earliest days of Computer Science, for the foundation paper of neural networks (McCulloch and Pitts 1943) was entitled *A Logical Calculus of the Ideas Immanent in Nervous Activity.* It had as its centerpiece the demonstration that the control box of any Turing machine (Turing 1936), the essential formalization of symbolic computation, could be implemented by a network of formal neurons. Moreover, the ideas of McCulloch and Pitts influenced John von Neumann and his colleagues when they defined the basic architecture of stored program computing, and von Neumann (e.g., 1951, 1956) remained intrigued with the biological dimension of computing, both in neural networks and self-reproduction.

Why, then, more than 50 years after 1943 do we need a book on *Computational Architectures Integrating Neural and Symbolic Processes*? I want to show that, in fact, we need many such books, and this is one of them.

One possible book on "Integrating Neural and Symbolic Processes" would be on Cognitive Neuroscience: it would use clinical data and brain imaging data to form a high-level view of the involvement of various brain regions in human symbolic activity, and would use single-cell activity recorded from animals engaged in analogous behaviors to suggest the neural networks underlying this involvement. Such a book would integrate the work of psychologists, neurologists, and neurophysiologists along with the work applying computational concepts to the analysis of biological neurons. The catch, of course, is that the "analogous behaviors" of animals are not very analogous at all when it comes to such symbolic activities as language and reasoning. Thus, the greatest successes in seeking the neural underpinnings of human behavior have come in areas such as vision, memory, and motor control where we can make neural network models of animal models of human capabilities, not in the area of high-level symbolic reasoning. And so we come to the nature and the importance of the present book: it is about high-level intelligent processes — a contribution to computer science in general, and to Artificial Intelligence in particular, rather than to neuroscience.

The work of Turing, McCulloch and Pitts, and von Neumann came together with the work of Norbert Wiener in the 1940s to create the field of *Cybernetics* (Wiener 1948), "the study of control and communication in the animal and the machine." Cybernetics was based on concepts like feedback and information, mixing McCulloch-Pitts neural networks with the engineers' theories of communication and control. The volume *Automata Studies*, published in 1956 and containing von Neumann's "Probabilistic logics and the synthesis of reliable organisms from unreliable components," was a major event in the development of the automata theory/neural networks component of cybernetics, yet in that very same year one of this volume's editors, John McCarthy, coined the term *Artificial Intelligence* (AI) at a meeting held at Dartmouth College. When in 1961 Marvin Minsky (a contributor to *Automata Studies* and at that time a colleague of McCarthy's at MIT) published the article "Steps toward artificial intelligence" which helped define the subject, cybernetic concepts and neural nets were very much part of his formulation of AI. Yet, sadly, AI came to be seen more and more in opposition to cybernetic concepts and neural networks. The use of logic was seen by many as the sine qua non of intelligence, and serial computation (careful search, one item at a time) was taken by many to be a virtue rather than a bottleneck. Although the study of neural networks and cybernetics continued through the 60s and 70s (the many excellent articles in such journals as *Kybernetik*, later *Biological Cybernetics*, attest to that), it tended to do so outside computer science departments.

(One aside which shows the interwoven nature of all this is that it was Warren McCulloch who brought Seymour Papert to MIT in 1963, and thus laid the basis for the book (Minsky and Papert, 1969) which many see as the major intellectual attack on neural networks. Actually, it provided excellent contributions to neural network theory, and anyone who is convinced that results on the limitations of simple perceptrons showed the inutility of neural networks should give up the use of computers since it is also known that simple programs without loops have limited capability!).

In the 1970s, then, most computer scientists working in AI outside such specialty areas as computer vision or robotics focused exclusively on symbolic representations of knowledge, using logic and a variety of more or less serial search strategies to address game-playing, problem-solving, and commonsense reasoning. However, the success of expert systems for many domains, such as medical diagnosis (Shortliffe 1976), showed the importance of probabilities or levels of confidence in weighing diverse evidence rather than adhering strictly to logical inference; while such contributions to distributed AI as the HEARSAY speech understanding system (Lesser et al. 1975) favored a computational metaphor based on interacting agents rather than serial processing, a metaphor fully expressed in Minsky's (1985) "conversion" to a theory of intelligence rooted in *The Society of Mind.* On the technological front, the development of VLSI made parallel computation a practical, indeed crucial, part of computer science. And the 1980s saw an immense resurgence of interest in neural networks, sparked in no small part by the appeal of Hopfield (1982) to physicists; the reception of the collections edited by Rumelhart and McClelland (1986) by cognitive psychologists and a new generation of AI workers; and by the adoption by technologists of neural networks as "universal approximators" with a wide range of applications; as well as by the work of many others, both "old- timers" and newcomers too numerous to mention.

And so at last we come to the 1990s and to the long postponed answer to the question "Why do we need a book on *Computational Architectures Integrating Neural and Symbolic Processes*?" The present book is in the domain of cognitive psychology and AI: seeking a computational model which can efficiently implement high-level "intelligent processes," rather than seeking to model the detailed neural processes of the human brain. Fifty years on we build on the legacies of Turing and McCulloch and Pitts, but much has happened in symbolic computation since Turing. Where he spoke of general effective procedures operating on a string of 0s and 1s, we have learned how to define hierarchical, symbolic structures - whether a search tree, a relational database, or an AI frame - which simplify the representation of data on a given domain and make the definition of operators far more transparent than would otherwise be possible. We have learned how to automate many of the processes of logical inference, and have seen the creation and recreation of vast structures of theoretical linguistics. And much has happened in neural networks since McCulloch and Pitts. Where they showed us how to translate logical formulas and state transitions into networks of "logical neurons," the emphasis has now switched to artificial networks whose *connection strengths* (thus the term *connectionism* used by many for the study of networks of "non-biological neurons") are subject to a variety of learning rules. We have learned how to exploit the parallelism of neural networks in a vast array of problem domains from vision to diagnosis, and in particular have developed powerful learning theories, both for self-organization of networks, and for their learning in response to supervision or reinforcement. We have found tasks for which a single neural network serves admirably, others for which an array of specialized networks serves best, and yet other for which a hybrid of neural networks and abstract symbol processors appears optimal. The present volume exemplifies each of these architectures in providing fresh approaches to many of the problems which, until a decade ago, had seemed securely and purely in the domain of symbolic processing alone: arithmetic, commonsense reasoning, story comprehension, and language processing — to convincingly demonstrate the power of integrating neural and symbolic processes.

REFERENCES

[1] Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational properties. *Proceedings of the National Academy of Sciences*, 79:2554–2558.

[2] Lesser, V.R., Fennel, R.D., Erman, L.D., and Reddy, D.R. (1975). Organization of the HEARSAY-II speech understanding system. *IEEE Trans. on Acoustics, Speech, and Signal Processing*, 23:11–23.

[3] McCulloch, W.S. and Pitts, W.H. (1943). A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophys.*, 5:115–133.

[4] Minsky, M.L. (1961) Steps toward artificial intelligence. Proc. *IRE*, 49:8–30.

[5] Minsky, M.L. (1985). *The Society of Mind.* Simon and Schuster, New York.

[6] Shortliffe, E.H. (1976). *Computer-Based Medical Consultations: MYCIN.* Elsevier, New York.

[7] Turing, A.M. (1936) On computable numbers with an application to the Entscheidungsproblem. *Proc. London Math. Soc. Ser. 2*, 42:230–265.

[8] von Neumann, J. (1951). The general and logical theory of automata. In L.A. Jeffress (Ed.), *Cerebral Mechanisms in Behavior: The Hixon Symposium.* Wiley, pp.1–32.

[9] von Neumann, J. (1956). Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E. Shannon and J. McCarthy (Eds.), *Automata Studies.* Princeton University Press, pp.43–98.

Michael A. Arbib
Center for Neural Engineering
University of Southern California
Los Angeles, CA

Preface

The focus of this book is on a currently emerging body of research — computational architectures integrating neural and symbolic processes. With the reemergence of neural networks in the 1980's and its emphasis on overcoming some of the limitations of symbolic AI, there is clearly a need to support some form of high-level symbolic processing in connectionist networks. As argued by many researchers, on both the symbolic AI and connectionist sides, many cognitive tasks, e.g., language understanding and commonsense reasoning, seem to require high-level symbolic capabilities. How these capabilities are realized in connectionist networks is a difficult question and it constitutes the focus of this book.

Although there has been a great deal of work in integrating neural and symbolic processes, both from a cognitive and/or applicational viewpoint, there has been relatively little effort in comparing, categorizing, and combining these fairly isolated approaches. Recently, there have been many new developments, some of which were reported at the *AAAI Workshop on Integrating Neural and Symbolic Processes (The Cognitive Dimension)* that was held in July, 1992, in association with the Eleventh National Conference of Artificial Intelligence; and those reported in the 1993 special issue of *Connection Science*, Vol.5, No.3-4. This body of work needs to be better understood, especially in terms of its architectural approaches.

The editors of this book intend to fill this void and address the underlying architectural aspects of this integration. In order to provide a basis for a deeper understanding of existing divergent approaches and provide insight for further developments in this field, the book presents (1) an examination of specific architectures (grouped together according to their approaches), their strengths and weaknesses, why they work, and what they predict, and (2) a critique/comparison of these approaches. The book will be of use to researchers, graduate students, and interested laymen, in areas such as cognitive science, artificial intelligence, computer science, cognitive psychology, and neurocomputing, in keeping up to date with the newest research trends. It can also serve as a comprehensive, in-depth introduction to this new emerging field. A unique feature of the book is a comprehensive bibliography at the end of the book.

Some of the questions addressed in the book are:

1. What architectural approaches exist and what are the relative advantages and/or disadvantages of each approach?
2. How cognitively plausible is each proposed approach?
3. Is there any commonality among various architectural approaches? Should we try to synthesize existing approaches? How do we synthesize these approaches?

   And also, more generically,
4. What processes are natural to do at the neural and symbolic level of description?
5. How do symbolic representation and connectionist learning schemes interact in integrated systems with different architectures?
6. What are the problems, difficulties and outstanding issues in integrating neural and symbolic processes?
7. What have we achieved so far by integrating neural and symbolic processes?

The body of the book starts with an introductory chapter, which comments on the current state of affairs and addresses what advances are necessary in order that continued progress be made. Following that, each particular architectural approach (i.e., localist, distributed, and integrated localist and distributed) is described by three chapters. The final two chapters compare some of the existing approaches. Note that we do not include work that is solely concerned with pure learning issues, neural networks for low-level processing, and pure engineering applications, since those topics are better covered by other existing books.

Finally, we wish to thank all the authors who contributed to this book.

Ron Sun
Lawrence Bookman

COMPUTATIONAL ARCHITECTURES INTEGRATING NEURAL AND SYMBOLIC PROCESSES

A PERSPECTIVE ON THE STATE OF THE ART

1

An Introduction: On Symbolic Processing in Neural Networks

RON SUN

Department of Computer Science
College of Engineering
The University of Alabama
Tuscaloosa, AL 35487

1 INTRODUCTION

Various forms of life have been existing on earth for hundreds of millions of years, and the long history has seen the development of life from single cell organisms to invertebrates, to vertebrates, and to humans, the truly intelligent beings. The biological organizations of various species, from the lowest to the highest, differ in their complexities and sizes. Such differences in internal complexity manifest in the differences in overt behaviors and intelligence. Generally speaking, organizational complexities of various species are proportionate with capabilities displayed by respective species. However, a gap seems to exist when one goes from high vertebrate animals to humans, in that a conscious, rational capacity is readily available to human beings, that does not seem to be present in any other animals, no matter how high they are on the evolutionary hierarchy. There is a qualitative difference. Yet, strange enough, there is no known qualitative difference between the biological make-up of human brains and animal brains. So the questions are: Where does the difference lie? What is the key to the emergence of rational thinking and intelligence?[1]

It is well known that human and animal brains are made up of "neural networks", or densely interconnected special kinds of cells, namely neurons, that possess simple "information processing" capabilities through the opening and

[1] The physical symbol hypothesis attempts to answer this question. However, it is deficient in several respects: it does not answer the question of how intelligent behaviors and their requisite physical symbols (according to the advocates of the theory) emerge from biological systems; it does not take into account subconceptual and intuitive thinking (see Smolensky [22] and Dreyfus & Dreyfus [5]) and does not address the question of how rational thinking can be coupled with such intuitive thinking; it ignores low-level processes (such as pattern recognition) and their interactions with high-level processes (Harnad [11]).

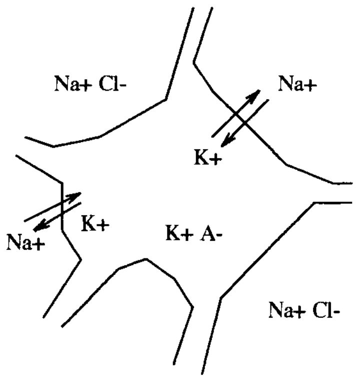


Figure 1 Diagram of a neuron with ion channels.

closing of membrane ion channels (and the enabling and disenabling of ion flows; see Figure 1) and through induced electrical spikes propagating from neuron to neuron (roughly speaking at least; for details, see, e.g., Kandel & Schwartz [13]). One question that naturally arises in this regard is how macro-level information processing, such as recognizing a face or bringing back a memory, can be accomplished with such micro-level "information processing" (i.e. channel opening and closing). A further question is how high-level conscious, rational thinking, such as drawing an inference or understanding a sentence, can emerge from networks of such low-level elements. The field of (artificial) neural networks seems to be addressing such problems, although the emphasis is on studying the capabilities of simplified and highly schematized artificial "neurons" (i.e. abstract mathematical models of simple processing elements), rather than exploring capabilities of biological neural networks (with the exception of a few groups of neuroscientists whose interests lie exclusively in information processing in biological neurons and in their interconnecting networks).

Some progress has been made to date in understanding low-level information processing (such as pattern recognition or associative memory) in neural networks, with highly homogeneous nodes and with regular connections among them (see, e.g., Figure 2 for such a model). The homogeneity and regularity enable mathematical and computational analyses of these models and thus facilitate advances in this field. Given the fact that preliminary understanding of neural networks for low-level information processing has been achieved (or at least is well on the way of being achieved), the other question, which is much more profound, immediately arises and cries out for answers: How can such

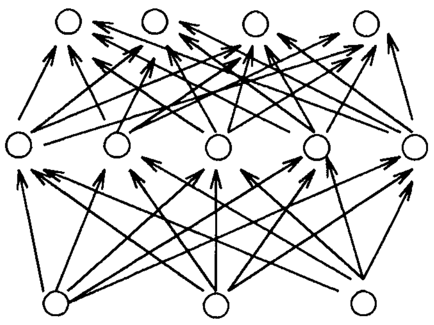


Figure 2 Diagram of a neural network with a regular interconnection pattern.

simple models capture high-level cognitive processes, such as rational thinking, natural language understanding, and logical inference? It is currently well understood (or presupposed, according to some people) that symbolic processing is capable of capturing a wide range (or all, according to some others) of rational thinking and intelligence; therefore it is of paramount importance to be able to incorporate symbolic processing in these models, to enable them to capture high-level cognitive processes. This is, however, especially difficult, because such processes are mostly discrete, structured, and systematic (see Fodor & Pylyshyn [8]), which are in sharp contrast with the characteristics of conventional connectionist models. The research being done under the banner of high-level connectionism and under the banner of hybrid systems, including the work described in the present book, attempt to answer exactly the above question, each in a partial and small way, by exploring the symbolic processing capability of neural network models. Note that, while the term "high-level connectionism" tends to describe cognitively motivated research, such as natural language processing (Sharkey & Reilly [21]) or commonsense reasoning (Sun [26]), the term "hybrid systems" tends to describe engineering-oriented approach toward incorporating symbolic processing or systems, such as fuzzy logic based neural networks (Bezdek [3]) or neural networks preprocessors coupled with symbolic planning (Hendler [12]). In my view, both schools are invaluable to the endeavor of investigation into how organizations of (artificial) neural networks can give rise to intelligence, thinking, and other high-level cognitive capacities. Thus I will not draw the distinction any further in the subsequent discussion.

In the following sections, I will take a brief look at high-level connectionist models incorporating symbolic processing, and discuss the issues and difficulties facing this field as well as possible future directions. Then I will present an overview of the chapters in the present book on the basis of the discussion.

2 BRIEF REVIEW

Let us look back briefly at some work in the short history of connectionist symbolic processing models. Incorporating symbolic processes in (artificial) neural networks is not a new idea at all. In as early as McCulloch & Pitts [15], it was shown that simple network models can perform some simple logical operations easily, and thus it was conjectured that human intelligence can be modeled with these simple logical operations (which contributed to the invention of digital computers and later, ironically, the enormous popularity of digital computers temporarily put an end to the fruitful research on neural networks).

The new connectionism of 1980's had been spawned mainly by work in cognitive modeling, including language learning and schemata processing (see, e.g., Rumelhart & McClelland [19]). However, the majority of work in this field shifted quickly to focus on low-level processes of vision, speech recognition, and sensory-motor control, as well as engineering applications, for these areas are particularly amenable to modeling with regular and homogeneous networks. Thus, high-level connectionist models (for modeling high-level cognitive processes) started to gain a separate identity, featuring a combination of neural network models and AI techniques in various ways.[2]

Work in the area of high-level connectionist models (which incorporate symbolic processing) includes Touretzky & Hinton [28], which first demonstrated that neural networks can implement symbolic production systems (in their simple form, as 3-tuples) and carry out corresponding symbolic reasoning. Work in this area also includes Barnden [2], which described a grid-form connectionist network for syllogistic reasoning incorporating the mental model approach. Work done by Dyer and associates (e.g., Dyer [6], Miikkulainen & Dyer [16] and Lange & Dyer [14]) focused on implementing symbolic AI ideas, especially Schankian constructs (such as scripts, schemas, and dynamic memory), in neural network models. The 1989 Cognitive Science Conference saw three papers on implementing rule-based reasoning in connectionist mod-

[2]Since it is common knowledge that it is difficult, if not impossible, to capture high-level cognitive processes with fully homogeneous connectionist networks, the majority of high-level connectionist models utilizes some alternative structures, such as localist models, as will be discussed later.

els: Ajjanagadde & Shastri [1], Lange & Dyer [14], and Sun [23], all of which enables neural networks to carry out rule(logic)-based reasoning in some flexible way and handle variable binding through either phase synchronization or sign propagation (see Appendix A for a listing of other relevant publications in this area).

On the other hand, some researchers explored the capability of neural network learning algorithms, and the distributed representation they produce, for symbolic processing. Pollack [17] and Sharkey [20] exploited the capability of distributed representation and showed how complex structures can be processed (to some extent) with such representations. There are also various systems for utilizing distributed representation to accomplish a variety of other tasks, ranging from simple similarity-matching to inheritance reasoning and to unification of logical terms (cf. Cottrell [29] and Sun [26]).

There have also been various proposals as to how to combine different types of models, such as combining symbolic models with connectionist networks (e.g., Bookman [4] and Hendler [12]) or combining localist and distributed representations (e.g., Sun [26]). The goal of such combinations is generally to enhance the symbolic processing power of respective models, by utilizing the synergy of different types of models. In this regard, there have been some initial successes for combined systems.

The *AAAI Workshop on Integrating Neural and Symbolic Processes (The Cognitive Dimension)*, which was held in July 1992, brought together work utilizing various approaches in developing symbolic processing connectionist models and explored how to compare, categorize and combine various isolated models and architectures in an effort to better understand these individual models, their interrelations, and the state of the art of this field overall. The work presented at the workshop was collected in [30]. The present book is the outgrowth of this workshop, with a slightly enlarged scope (in that we are no longer exclusively concerned with the "cognitive dimension").

3 EXISTING APPROACHES

Now let us examine what various symbolic processing connectionist models (such as those mentioned above) have in common; in other words, we want to categorize various models in terms of a few types in order to see emerging trends and to identify issues and problems.

The existing architectural approaches in integrating neural and symbolic processing (that is, with respect to the internal organizations of and the representations used in respective models) can be divided into the following four categories:

- Developing specialized, structured, localist networks for symbolic processing.
- Performing symbolic processing in distributed neural networks (in a holistic way).
- Combining separate symbolic and neural network modules.
- Using neural networks as basic elements in symbolic architectures (the embedded approach).

The first approach above uses individual nodes to represent individual concepts (and hence the term "localist"), and the connections between nodes directly reflect the linkage between the corresponding concepts. Such architectures amount to directly map symbolic structures onto connectionist network structures and in the process, massively parallel systems result. This approach was formerly advocated by Feldman & Ballard [7]. (See also Chapters 3 and 4.)

The second architectural approach represents concepts as distributed patterns (of activations) across a large number of nodes in a network (or a certain part of a network, i.e., a "layer"). This approach is thus connectionist in its purest form. Some connectionists believe the strong connectionist thesis that simple networks, trained with corresponding learning algorithms (such as the backpropagation learning algorithm), can perform the functional equivalent of symbolic processing in a holistic and functional way. This strong thesis is in fact quite controversial. However, most connectionists do believe that symbolic processing can be accomplished in a holistic and functional way to some large extent. Thus, this type of architectures remains an active subject of connectionist symbolic processing research. (See Chapters 5, 6 and 7.)

The third architectural approach is the juxtaposition of two or more components, or "modules". Each of these modules uses a different type of representation, ranging from purely symbolic systems to distributed connectionist systems, but together they accomplish a wide range of processes. Currently, there are a variety of ways of organizing these modules, depending on the types of modules used and the tasks to be accomplished. For example, the loosely coupled organization allows communication through an interface that connects various modules and moderates information flows; such an organization is prevalent in and good for (i.e., facilitating the development of) application systems. The tightly coupled organization uses a variety of different channels for communication, and thus allows closer interaction between modules. The completely integrated organization has so many connections between various modules that the distinction between modules almost vanishes, although different representations are used in these different modules. However, in order to be completely integrated, usually only different *connectionist* representations can be used. This type of architectures is currently a highly active area of research, for both application-oriented and theoretically-motivated work. (See Chapters 8, 9 and 10.)

The forth approach utilizes basically a symbolic architecture overall, such as a semantic network or a parallel production system, but instead of using symbolic components, small scale neural networks are used in their places to enable parallel, fault-tolerant computation that is capable of partial matching.

It should be cautioned that these types are not as clear-cut as they seem to be from the foregoing discussion. As a matter of fact, they are interrelated and often they are mixed together in a model (that is, one model involves more than one architectural approach). In addition, these approaches are still in the process of evolving. Therefore, any classification scheme should be taken with a grain of salt.

4 Issues and Difficulties

There are many research issues that need to be addressed, in order to advance this field of study, as well as, more broadly speaking, to better understand the nature of intelligence and cognition. These issues can be addressed in technical terms (focusing on techniques) as well as in biological terms (in relation to biological systems). This section takes a brief look at several of these issues from a technical perspective.

Can purely connectionist systems account for all kinds of cognitive processes and model all kinds of intelligent behaviors? Purely connectionist systems, such as backpropagation networks, are known to be able to perform certain types of symbolic processing; the question is how far we can push this type of systems and how much symbolic processing that is necessary for high-level cognitive processes this type of systems can ultimately account for. Some researchers keep on pushing the frontier in this direction, and try to prove the sufficiency, or at least the dominant role, of such processes in modeling cognition and intelligent behaviors (recall the strong connectionist thesis mentioned earlier). So far, some progress has been made, and more symbolic processing tasks necessary for modeling high-level cognitive processes (such as structure-sensitive operations in distributed representation) are being performed in these systems (see, e.g., Chapter 7). In utilizing such systems for performing various symbolic processing tasks (such as embedding part-whole hierarchies, language induction, and implementing production systems), better understanding has been achieved of this type of connectionist models. However, it is clear that not all kinds of cognitive processes can be captured in such processes (at least not yet). While the aforementioned researchers are continuing their work in this direction, other researchers look into other means for accomplishing symbolic processing tasks (as in Chapters 3 and 8). Generally, structured localist networks and semantic-network-like models (i.e., links represent some meaningful connection, such as *is-a* or *part-of*, between concepts each of which is represented by an individual node in a network) are used to replace or supplement purely connectionist forms. Whether such structured models can eventually be subsumed by purely connectionist models or whether they will eventually be an alternative form, or even the prevailing form, of connectionist systems (with purely connectionist models as special cases) is yet to be seen. In any event, some sort of convergence of the two approaches seems to be necessary, in that structured localist systems need the learning capability usually associated with purely connectionist systems and, conversely, connectionist systems need to be able to represent complex structures somehow.

Do we need a specialized connectionist system for each kind of cognitive processes? In other words, one system or many? It has been an acute problem that in symbolic AI, for each type of tasks, a set of specialized mechanisms are developed, which are not necessarily related to any other mechanisms for any other tasks. This practice creates an abundance of specialized mechanisms and limited models, but also creates a lack of uniformity and coherence. It should be noted that it is usually not the case that once one has all the components, a whole necessarily follows. It has been a tremendous problem for traditional AI to form a coherent system incorporating various processes; even if a kludge (an ad hoc collection of disparate mechanisms) can be worked out, what kinds of understanding can such a system provide us, beside the fact that it is a kludge? It is, in some sense, an issue of long-term progress vs. short-term expediency, since in the short term, within some limited task domains, a kludge may indeed work better, but in the long term, fundamental principles, elegantly and succinctly expressed, are much more desirable and much more important. The same problem has started to emerge in connectionist research: when more and more specialized connectionist models are proposed, a unifying theory, explanation, or model is what is most needed. Some researchers are beginning to address this problem, by including in a single model a range of capabilities and by trying to propose a uniform explanation of a variety of processes (e.g., with a uniform mechanism encompassing many processes; cf. Sun [26]).

How can more powerful learning algorithms be developed that can acquire complex symbolic representation in connectionist systems, including structured localist systems? This is an important issue, in that simple learning algorithms that build up functional mappings in typical neural network models are insufficient for symbolic processing connectionist networks, because of the discrete and discontinuous nature of symbolic processes and because of the systematicity of such processes. Newer and more powerful learning algorithms are needed that can extract symbolic structures from data and/or through interaction with environments. As a minimum requirement, for example, such learning algorithms must be able to handle embedding relations among symbolic structures (Sun [27]) and combinational composition of symbolic structures (Fodor & Pylyshyn [8]). However, unfortunately, currently there is no learning algorithm that adequately meet these needs.[3] Therefore, most of the existing work on integrating neural and symbolic processes focuses on representational aspects rather than learning-related aspects and the interplay between learning and representation, although such interplay is one main appeal of the original connectionist paradigm (cf. Hanson and Barr [10]).

One practical difficulty in developing learning algorithms for complex symbolic structures (such as those in Lange & Dyer [14] or in Sun [25]) lies in the fact that neural network learning algorithms tend to be simple, numerical, and structurally uniform (with the same, or similar, operation being applied to all the nodes in an entire network), which is in direct contrast to the characteristics of symbolic representation (as in, e.g., Sun [25]), where irregular, content-dependent connections and discrete, individuated activation functions are commonplace. Therefore, some new kinds of learning algorithms are needed for symbolic processing connectionist models; such algorithms, I believe, should somehow incorporate some symbolic methods, as more powerful learning algorithms will result from such incorporation.

[3] I did not mention the backpropagation networks using distributed representation in the preceding discussion. Although such networks can accomplish some of the aforementioned processing, there are various difficulties with them, such as long training time, inaccuracy in symbolic mappings, and the inadequacy in structure-sensitive operations (although some of them can be carried out holistically).

How can the distinction between conceptual and subconceptual processes be profitably explored in symbolic processing connectionist models? The distinction between conceptual and subconceptual processes has been argued for by, for example, Smolensky [22],[4] and it seems intuitively very appealing as a way of capturing the two differing styles of thinking and useful in a theoretical understanding of the role of connectionist models in cognitive modeling. However, this distinction needs to be qualified and made clear and precise; connectionist models, especially high-level connectionist models, provide the hope that this idea can be studied through computational modeling and experimentation, with integrated connectionist systems in which both symbolic and subsymbolic processes are captured (in either a separate or a mixed manner). Through such experiments, a mechanistic explanation of the distinction between conceptual and subconceptual processes may result, for the benefit of an enhanced theoretical understanding. On the other hand, future advances in connectionism require the understanding of the theoretical notions and ideas in order to come up with principled solutions for problems arising in cognitive modeling (and in other areas as well): for example, on the one side, conscious rule application [9], explanation generation, and rule manipulation and modification, and on the other side, intuitive, holistic, and non-verbal (tacit) reasoning, and also their interaction in cognition (cf. Chapter 8). This mutual dependency of theories and computational models is not uncommon in AI and cognitive science, but in this case such dependency clearly needs serious attention and devoted efforts to explore.

I believe that in the dichotomy of conceptual and subconceptual processes may lie the key to achieving the fundamental understanding of the architecture of cognition and intelligence. Specifically, I believe that conceptual processes can be best captured by symbolic processes (in purely symbolic systems or in localist connectionist systems), while subconceptual processes can somehow be modeled by the (artificial) neural networks and their variants, as discussed by Smolensky [22]. In light of the above, it is highly desirable for connectionist researchers to try to utilize the dichotomy in their work and place their models in a proper place; it is also important for connectionist researchers to address the problem of modeling both types of processes (conceptual and subconceptual) in an integrated architecture, in order to strive for a better un-

[4]The conceptual processes handle knowledge that possesses the following characteristics: (1) public access, (2) reliability, and (3) formality. The appropriateness of modeling such knowledge by symbolic processes has been argued for by many and seems self-evident. But on the other hand, there are different kinds of knowledge, such as skill, intuition, individual knowledge, and so on, that are not expressible in linguistic forms and do not conform to the three criteria prescribed above. They constitute subconceptual processes. It seems futile to model such knowledge in symbolic forms.

derstanding of the overall cognitive architecture. So far, unfortunately, there are only a handful of models directly confronting the problem, probably due to the lack of experimental understanding regarding this issue and the lack of methodologies for approaching the problem, which in turn can probably be attributed to the long history of the dominance by rationalistic philosophies that ignore the intuitive, subconceptual, and subconscious side of cognition. We need to overcome such philosophical and methodological obstacles in order to make more fundamental progress. The effort may reasonably be expected to produce profound results. Hopefully, models addressing this issue may help to shed some light on the puzzle concerning the difference between human and animal intelligence (or, conceptual processes vs. subconceptual processes and rationality vs. association; see Section 1 of this chapter), in that a mechanistic explanation of the distinction may be produced based on the same basic building material, i.e., artificial neurons.

5 Future Directions, Or Where Should We Go From Here?

Although the outlook of this field is still murky, there are indeed several trends discernible at this point in time. Whether they will come to full fruition in the future, which, as usual, depends on the progress of the whole field and also depends on the interplay of these trends, is yet to be seen.

First of all, connectionist systems being developed are becoming increasingly more complex. They tend to encompass more and more functional aspects, and more and more varieties of different techniques. For example, there is little work now dealing exclusively with the variable binding problem, or focusing only on word sense disambiguation. Instead, solutions for the variable binding problem are being integrated into larger systems of reasoning, possibly along with approximate matching, multiple instantiation, and type hierarchies (implicit or explicit). By the same token, models for word sense disambiguation are becoming part of systems for story comprehension or other higher-level tasks. This, on the one hand, indicates in some way the maturation of the field, in that simple, partial models and techniques have been explored to a point that some syntheses are becoming possible or even necessary. This maturation of the field allows the development of systems on a scale roughly comparable to traditional "symbolic" systems; such larger-scale systems are more useful in artificial intelligence and cognitive research, and will definitely generate more impact for AI and cognitive science in the years to come.

On the other hand, the increasing complexity of connectionist systems entails the necessity of *integration* instead of mere combination, due to the need to manage the complexity and to avoid ad hoc-ness, as alluded to before. There are some promising new developments in this respect (see, e.g., Chapter 8), but more efforts are certainly needed to further the development.

Another trend is the flourishing of application-oriented system development, as evident from recent neural networks and AI conferences. Applications of connectionist symbolic processing systems to real world problems are not only beneficial to application domains, but are also important to the future development of the field, in that such applications can provide new incentives, new motivations, and new problems for theoretical studies. Both the theoretical research and the practical development are worth emphasizing.

Yet another direction that is of great importance to the future of this field is the development of a clear and concise theoretical (and/or mathematical) framework, one that supersedes existing models and provides directions for future advances, in ways similar to what the backpropagation algorithm did for the early connectionist research, or what Maxwell's equation did for the study of electromagnetism. The advances in model building and the resulting diversity in perspectives, approaches, and techniques in the area of connectionist symbolic processing models call for serious theoretical treatments to clarify the existing ideas and to generate new thinking. A solid theoretical foundation is what is most needed for this field. The present book itself is an attempt in this direction, in that it provides a framework for the field, although it is not focused on the theoretical issues per se.

6 OVERVIEW OF THE CHAPTERS

The rest of the chapters in this book can be divided into four parts. Each of the first three parts covers a different architectural approach, in accordance with the foregoing discussion and classification (see Section 3 of this chapter): Part 1, which includes Chapters 2, 3, and 4, covers localist architectures; Part 2, which includes Chapters 5, 6, and 7, covers distributed architectures; Part 3, which includes Chapters 8, 9, and 10, covers combined architectures. The last part of the book, which includes Chapters 11 and 12, is a set of commentaries critiquing work in this field, including those reported in this book.

In Part 1, Chapter 2 (by John Barnden) describes symbol processing in a "transiently" localist connectionist model, with representations being constructed on the fly. The system performs syllogistic reasoning through the manipulation of mental models (as proposed by Philip Johnson-Laird). Some unique localist representational techniques are developed; the techniques predispose the system towards random instead of pre-ordered sequencing of subtasks, and towards associative linking of symbolic structures as opposed to explicit linking constructs. The system is elaborate and complex — it is one of the most complex symbolic processing problems to be tackled in connectionism.

Chapter 3 (by Trent Lange) describes a structured localist connectionist model capable of high-level inferencing with variable binding and rule application. In his model, variable binding is handled by distinct activation patterns that uniquely identify the concept bound to a variable. Rules are pre-wired into the structure of the network, resulting in a semantic network like system. Based on such a model, he further develops a system for integrating language understanding and episodic memory retrieval with spreading activation.

Chapter 4 (by Chris Lacher and K. D. Nguyen) presents a way of combining expert systems and neural network learning, resulting in a localist network model, which is termed "expert networks" by the authors. This method has the advantage of being able to learn and adapt, unlike traditional expert systems, and the advantage of being able to utilize pre-wired structures, in addition to homogeneous neural network connectivity patterns. The authors also introduces ways of implementing each node in "expert networks" with a small scale neural network, which is related to the embedded approach mentioned earlier (see also [25]).

In Part 2, Chapter 5 (by Risto Miikkulainen) presents a distributed connectionist model for processing sentences with recursive relative clauses. The architecture deals with the tasks of segmenting word sequence into clauses, forming case-role representations, and keeping track of recursive embeddings. The model is a purely distributed connectionist model, and has many usual properties of such systems such as generalization, graceful memory degradation, and statistical constraint induction.

Chapter 6 (by David Noelle and Gary Cottrell) presents an approach for distributed connectionist networks to "learn by being told". While many learning algorithms have been proposed which allow connectionist models to modify their representation based on examples and statistical correlations in them, this approach allows input directives to influence the behavior of a network directly. Given the difficulties of connectionist learning techniques in addressing symbolic processing tasks, such an approach may go a long way to shed new light on connectionist symbolic processing models. The authors also examine a distributed connectionist network that performs a reverse task: generating linguistic descriptions of time-varying scenes.

Chapter 7 (by Noel E. Sharkey and Stuart Jackson) analyzes theories of distributed connectionist representation. It challenges a key assumption of such theories: that the precise distances between distributed representations in the hidden layer of a backpropagation network reflect systematic semantic and/or structural similarity relations. A detailed argument is provided that utilizes a simple decision space technique, demonstrating that this assumption is not warranted except under special circumstances. The authors claim that the computational role of a distributed representation may be separated from specific distance relations.

In Part 3, Chapter 8 (by Ron Sun) presents a connectionist architecture that consists of two levels: one is an inference network with nodes representing concepts and links representing rules connecting concepts (i.e., with localist representation), and the other is a microfeature based replica of the first level (with distributed representation). Based on the interaction between the concept nodes and microfeature nodes in the architecture, inferences are facilitated and knowledge not explicitly encoded in a system can be deduced via a mixture of similarity matching and rule application. The model is for structuring knowledge in vague and continuous domains where similarity plays a large role in performing plausible inferences. The architecture is able to take account of many important desiderata of plausible reasoning, and produces sensible conclusions accordingly.

Chapter 9 (by Larry Bookman) presents a connectionist model that supports both structured representations and non-structured representations in which knowledge is encoded automatically using information-theoretic methods. A two-tier structure is proposed to encode such knowledge: a relational tier (network) that represents a set of explicit conceptual relations, and an associational tier (network) that encodes the associational or nonstructured knowledge. This model supports two complementary views of text comprehension: a "coarse-grain" view, that utilizes explicit semantic relationships to reason about the "meaning" of a text; and a "fine-grain" view, that explores details of interaction between context and background knowledge.

Chapter 10 (by Charles Lin and Jim Hendler) presents an application of a hybrid system. A hybrid system shell is developed and used in the task of detecting certain patterns in the sensor traces of ballistic firings. The hybrid system contains both connectionist (neural network) components and expert system components. The chapter explains how the expert system can be used to add to the ability of the neural network, by using expert domain knowledge.

In Part 4, Chapter 11 (by Vasant Honavar) attempts to explore questions regarding fundamental similarities and differences between symbolic systems and connectionist systems. A historical examination of such questions leads to the conclusion that the two paradigms offer formally equivalent but practically different models, and their integration is useful when various design alternatives are fully explored.

Chapter 12 (by Michael Dyer) reviews the state of the art of natural language processing with connectionist models. Specifically, he examines the following issues in these models: (1) the creation and propagation of dynamic bindings, (2) the manipulation of recursive, constituent structures, (3) the acquisition and access of lexical, semantic, and episodic memories, (4) the control of multiple learning/processing modules, and (5) the "grounding" of basic-level language constructs in perceptual/motor experiences. The chapter indicates the current strengths and weaknesses of various approaches.

7 SUMMARY

The present chapter presents an introduction to the field of connectionist symbolic processing models, reviewing its development, discussing various approaches, highlighting some issues, and pointing out possible future directions. Overall, in the field of connectionist symbolic processing models, the ground has been broken and some foundations have been laid down. However, to complete the skyscraper that we are aiming for on the basis of what we have so far, a lot more hard work is still needed; the process of advancing connectionist symbolic processing models to the stage in which these models serve as a dominant paradigm for AI and cognitive research still requires much ingenuity to come up with workable new ideas and plausible innovative designs. I believe that the enterprise of connectionist symbolic processing models holds great promise for artificial intelligence and cognitive research, and may even help to shed light on some deep philosophical questions; thus it is well worth pursuing.

ACKNOWLEDGEMENTS

This work is supported in part by the RGC grant #1640 from the University of Alabama.

REFERENCES

[1] V. Ajjanagadde and L. Shastri, Efficient Inference with Multi-place Predicates and Variables in a Connectionist System, *Proc.11th Cognitive Science Society Conference*, pp.396-403, Lawrence Erlbaum Associates, Hillsdale, NJ. 1989

[2] J. Barnden, The right of free association: relative-position encoding for connectionist data structures, *Proc.10th Conference of Cognitive Science Society*, pp.503-509, Lawrence Erlbaum Associates, Hillsdale, NJ. 1988

[3] J. Bezdek, (ed.) *IEEE Transaction on Neural Networks*, special issue on fuzzy neural networks. 1992.

[4] L.A. Bookman, *Trajectories through knowledge space: A Dynamic Framework for Comprehension*, Kluwer Academic Publishers, Norwell, MA 1994.

[5] H. Dreyfus and S. Dreyfus, *Mind Over Machine*, The Free Press, New York, NY. 1987

[6] M. G. Dyer, Distributed symbol formation and processing in connectionist networks, *Journal of Expt. Theor. Artificial Intelligence*. 2, pp.215-239, 1990.

[7] J. Feldman and D. Ballard, Connectionist models and their properties, *Cogni tive Science*, pp.205-254, July 1982

[8] J. A. Fodor and Z. W. Pylyshyn, Connectionism and cognitive architecture: A critical analysis, *Cognition*. 28, pp. 3-71. 1988.

[9] R. Hadley, Connectionism, rule following, and symbolic manipulation, *Proc.of AAAI-90*, Vol.2, pp.579-586. Morgan Kaufman, San Mateo, CA. 1990.

[10] S. Hanson and D. Barr, What does the connectionist model learn: learning and representation in connectionist models, *Behavior and Brain Sciences*, 13(3), pp.1-54, 1990

[11] S. Harnad, The symbol grounding problem. *Physica D*, 42(1-3):335–346. 1990.

[12] J. Hendler, Integrating neural and expert reasoning: an example. In: *Proc. of AISB-92*. pp.109-116. 1992.

[13] E. Kandel & J. Schwartz, *Principles of Neural Science*, Elsevier, New York, NY. 1984.

[14] T. Lange and M. Dyer, Frame selection in a connectionist model, *Proc.11th Cognitive Science Conference*. pp. 706-713, Lawrence Erlbaum Associates, Hillsdale, NJ. 1989

[15] W. McCulloch and W. Pitts, A Logical Calculus of the Ideas Immanent in Nervous Activity, *Bull. Math. Biophy.*, 1943

[16] R. Miikkulainen and M. G. Dyer. Natural Language Processing with Modular PDP Networks and Distributed Lexicon. *Cognitive Science*. 15, 3, 1991.

[17] J. Pollack, Recursive distributed representations. *Artificial Intelligence*, 46:77–105. 1990.

[18] F. Rosenblatt, *Principles of Neurodynamics*. Spartan Books, New York, NY. 1962.

[19] D. E. Rumelhart, J. L. McClelland, & the PDP Research Group. *Parallel distributed processing: Explorations in the microstructure of cognition*. Cambridge, MA: Bradford Books. 1986.

[20] N. E. Sharkey, Connectionist representation techniques. *AI Review*, (5):142–167. 1991.

[21] N. Sharkey, & R. Reilly. (eds.) *Connectionist Approaches to Natural Language Understanding*. Hillsdale, NJ: Lawrence Erlbaum Assoc. 1991.

[22] P. Smolensky, On the proper treatment of connectionism, *Behavioral and Brain Sciences*, 11, pp.1-43, 1988

[23] R. Sun, A discrete neural network model for conceptual representation and reasoning. In: *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum. 1989.

[24] R. Sun, The Discrete Neuronal Model and the Probabilistic Discrete Neuronal Model, in: B.Soucek (ed.) *Neural and Intelligent Systems Integration*, John Wiley and Sons, New York, NY. pp.161-178. 1991.

[25] R. Sun, On Variable Binding in Connectionist Networks, *Connection Science*, Vol.4, No.2, pp.93-124. 1992.

[26] R. Sun, *Integrating Rules and Connectionism for Robust Commonsense Reasoning*, John Wiley and Sons, New York, NY. 1994.

[27] R. Sun, On Schemas, Logics, and Neural Assemblies, *Applied Intelligence*, special issue on high level connectionist models, 1994.

[28] D. Touretzky and G. Hinton, Symbols among neurons, *Proc.9th IJCAI*, pp.238-243, Morgan Kaufman, San Mateo, CA. 1987.

[29] G. Cottrell, Parallelism in inheritance hierarchies with exceptions, *Proc.9th IJCAI*, pp.194-202, San Mateo, CA: Morgan Kaufman, 1985.

[30] R. Sun, L. Bookman, and S. Shekhar, (eds.) *The Working Notes of the AAAI Workshop on Integrating Neural and Symbolic Processes*, American Association for Artificial Intelligence, Menlo Park, CA. 1992.

PART I

LOCALIST ARCHITECTURES

Part I: Localist Architectures

- Chapter 2 (by John Barnden) describes a localist connectionist model with dynamically constructed representations for performing syllogistic reasoning.
- Chapter 3 (by Trent Lange) describes a structured localist connectionist model capable of high-level inferencing with variable binding.
- Chapter 4 (by Chris Lacher and Ky Nguyen) presents a method for combining expert systems and neural network learning.

2

Complex Symbol-Processing in Conposit, A Transiently Localist Connectionist Architecture

JOHN A. BARNDEN

Computing Research Laboratory and Computer Science Department
New Mexico State University
Las Cruces, New Mexico 88003

1 INTRODUCTION

Two unusual primitives for the structuring of symbolic information in connectionist systems were discussed in [9]. The primitives are called *Relative-Position Encoding* (RPE) and *Pattern-Similarity Association* (PSA). The present article shows that the primitives are powerful and convenient for effecting cognitively sophisticated connectionist symbol processing. Specifically, it shows how RPE and PSA are used in a connectionist implementation of Johnson-Laird's mental model theory of syllogistic reasoning [23] [24] [25]. The symbol processing achieved is therefore at the level of complexity to be found in existing, detailed information-processing theories in cognitive psychology. This system is called Conposit/SYLL, but for brevity it will often be referred to here as Conposit. To be exact, Conposit is a general framework for implementing rule-based systems in connectionism, and Conposit/SYLL is just one instance of it. (The name "Conposit" is derived from "CONnectionist POSITional encoding." Conposit/SYLL is a major extension beyond the preliminary version described in [2]).

The Johnson-Laird theory is used here merely as a case study in the application of the Conposit framework. In particular, it is not important here whether Johnson-Laird's theory is psychologically correct; even if it is correct, it is not important whether Conposit is anywhere near the actual implementation of the theory in the brain's neural circuitry. The reason for choosing to implement the Johnson-Laird theory is that computationally it has been relatively completely specified in the psychological literature, and presents a major implementational challenge to high-level connectionism. The nature of the challenge will be clarified later, but, briefly, it arises from the mutability, multiplicity, and diversity of the working memory structures that need to be set up, and the elaborateness of the sequences of modifications to them that should take place during reasoning. Conposit copes easily with the challenge.

The plan of the paper is as follows. Section 2 summarizes the Johnson-Laird theory of syllogistic reasoning, and explains the nature of the challenge it presents to connectionism. Section 3 explains how the Johnson-Laird theory is implemented in Conposit. The section is at a level a little above that of connectionist circuitry. Section 4 sketches the mapping of this level down to connectionist circuitry. Section 5 shows how the challenge described in section 2 is met by Conposit. This section includes a discussion of variable binding. Section 6 summarizes two simulation runs of Conposit/SYLL. Section 7 comments briefly on some other theoretical contributions of the Conposit framework. Section 8 concludes.

The PSA and RPE primitives make Conposit distinctly different in flavor from other connectionist systems that do symbolic processing (see, e.g., Bookman, this volume, Lange, this volume, Sun, this volume, and [18] [31] [32] [37].) However, the PSA technique in Conposit is closely related to Shastri and Ajjanagadde's use of synchrony for binding [31] (see [8] and [9] for discussion), and somewhat less so to Lange's use of signatures. A form of RPE technique is used in [15].

2 THE JOHNSON-LAIRD THEORY AND ITS CHALLENGES

2.1 SUMMARY OF THE JOHNSON-LAIRD THEORY

This subsection summarizes the mental model theory of syllogistic reasoning as presented in [23] and [24], making reference also to later developments in [25]. An example of a syllogism is

```
Some of the artists are beekeepers.
All the beekeepers are chemists.
[Therefore:]  Some of the artists are chemists.
```

In general, a syllogism is an argument, expressed in natural language, consisting of two *premises* that are followed by a *conclusion*. Each of the three propositions states some relationship between two sets of entities. The first proposition is about two sets A and B, the second is about sets B and C, and the third about A and C. The relationship is either a subset relationship (expressed as "all the X are Y"), an intersection relationship ("some of the X are Y"), a negated-subset relationship ("some of the X are not Y"), or a negated-intersection relationship ("none of the X are Y"). In each of the three propositions, the two sets can appear either way round. For example, the second premise could be either "all the B are C" or "all the C are B." Each of the three sets is assumed to be non-empty (in most of Johnson-Laird's work).

Of course, the actual sets chosen for a syllogism make no difference in a logical sense.[1] Also, the switching of the two sets in a premise may make no logical difference: "some of the A are B" and "some of the B are A" are logically equivalent, as are their negations, "none of the A are B" and "none of the B are A." However, the Johnson-Laird theory does count the equivalent forms as different premises. Again, the ordering of the two premises with respect to each other makes no logical difference, but nevertheless the ordering is taken as significant psychologically.

For a given pair of premises it may be impossible to complete the syllogism by means of a valid conclusion that relates sets A and C by one of the four relationships above. For example, from "some of the A are B" and "some of the B are C" no valid conclusion can be drawn. On the other hand, for a given pair of premises there may be more than one correct conclusion. For one thing, a conclusion can be replaced by an equivalent proposition (cf. previous paragraph). But there can also be non-equivalent alternatives: for instance, the proposition "some of the A are C" is a correct deduction from "all the A are B" and "all the B are C", granted the existence presupposition noted above. However, a stronger conclusion is that "all the A are C". The Johnson-Laird theory tries to generate "all the A are C" before "some of the A are C," and similarly "none of the A are C" before "some of the A are not C," so as to be maximally informative. Also, there is a processing order effect that helps determines whether the model tries to generate a conclusion in A–C order or in C–A order first, depending on the set-ordering in the premises.

The theory has it that people syllogize by constructing one or more "mental models" conforming to the premises, and then trying to generate conclusions from the models. One possible mental model for the above syllogism can be drawn as follows:

[1] This is not to say that they make no difference to a human reasoner — cf. the experimental results in [27] and [28].

MENTAL MODEL MM 2-1

```
(a)       (b)   =   (c)
 a    =    b    =    c
 a    =    b    =    c
                    (c)
```

A mental model is a data structure containing atomic *tokens* (shown by the letters) and *identity links* between tokens (shown by the equality signs). The model contains arbitrarily selected numbers of tokens standing for artists, beekeepers, or chemists (the occurrences of 'a', 'b' and 'c'). Because of the first premise of the syllogism, namely the proposition that some of the artists are beekeepers, an arbitrarily selected non-empty subset of the artist tokens is related by identity links to some beekeeper tokens. Similarly, because of the second premise, namely that all of the beekeepers are chemists, all beekeeper tokens are similarly linked to chemist tokens. The parentheses indicate that the enclosed tokens are declared as being optional. Distinct tokens not related directly or indirectly by identity links are taken to represent different individuals. The tentative conclusion that some artists are chemists would arise from the syllogizer noticing that some artist tokens are linked by chains of identity links to chemist tokens. Notice that a mental model can contain redundancy — the second line of MM 2-1 is not necessary. It does no harm apart from possibly slowing the processing.

A mental model thus serves as a highly abstract description of what may be viewed as an "example situation" conforming to the premises of the natural language syllogism. More precisely, what is described by MM 2-1 is really several example situations, differing on which of the optional individuals are actually present. Naturally, the tentative conclusion read off from a model might merely be an artifact of the particular example(s) it describes. Johnson-Laird therefore postulates that the system attempts to construct *several* different models conforming to the premises, in an attempt to falsify any particular tentative conclusion. The attempted-falsification process would fail in our example, but would succeed (if pushed to completion) in the variant example obtained by changing the second premise to "*some* of the beekeepers are chemists". This is because in choosing beekeeper tokens to link to chemist tokens, it might happen that none of the chosen beekeeper tokens are linked to artist tokens.

The process of building an initial model that conforms to the premises has two parts: creating a model conforming to the first premise, and then extending this by using the second premise to form a model of both premises. In the syllogism displayed above, the first premise would lead to a model containing just a and b tokens — perhaps the a and b portion of MM 2-1. This model would then be extended to produce a model involving chemist tokens. A possible result is MM 2-1.

A negative premise is handled in [24] by dividing the token set up into disjoint subsets by means of a "negative barrier." Conposit does away with the barriers, as does the later formulation of the theory in [25].

2.2 ADJUSTMENTS TO THE JOHNSON-LAIRD THEORY

The system to be presented embodies an adjusted, somewhat simplified form of the Johnson-Laird theory, while still having the same logical power. This section summarizes the adjustments. See [2] for a fuller discussion, with justifications. Some of the adjustments to the theory are in areas where Johnson-Laird has been unsure, not entirely consistent, or unclear. Also, some of the major changes actually make no difference from the standpoint of psychological predictions or experiment, since they are in aspects of the theory that are not appealed to in the presentations of the predictions and experiments. In any case, the version implemented in Conposit preserves the interesting challenges.

The key ideas of mental models, mental model construction from premises, derivation of tentative conclusions from mental models, and attempted falsification of tentative conclusions through construction of further models have been preserved in Conposit. On the other hand, some of the detail of the structure of models has been adjusted, and the method for exploring the space of mental models has been radically modified. The model-structure adjustments are natural ones that are suggested by the way Conposit works and make good sense in the terms of mental model theory as such.

The adjustments fall into two broad classes: representational and procedural. They will be considered in that order. The procedural ones partly follow from the representational ones.

Name Sharing instead of Identity Links

Johnson-Laird and Bara ([24], especially p. 28) imply that a line like

```
a = b = c
```

in a model corresponds to an ordered list of items in their computer program embodying the theory. Also, the left-right ordering in such a line is significant in the processing details of the *theory* as such. One of our adjustments, however, is to replace identity links by *sharing of token names*. Thus, instead of

```
a = b
```

the following *unordered set* of tokens is used:

$$\mathtt{A}:x \qquad \mathtt{B}:x$$

which could just as well be written down as:

$$\mathtt{B}:x \qquad \mathtt{A}:x$$

The x is some arbitrary label that uniquely names the individual represented by the token. Thus, the token set shown represents a single individual that is in both set A and set B. On the other hand, the token set

$$\mathtt{A}:x \qquad \mathtt{B}:y$$

represents two distinct individuals, since x and y are different names. Altogether, then, model MM 2-1 now takes the following form:

MODIFIED MENTAL MODEL MM 2-2

$\mathtt{A}:x_1$	$\mathtt{B}:x_1$	$\mathtt{C}:x_1$
$\mathtt{A}:x_2$	$\mathtt{B}:x_2$	$\mathtt{C}:x_2$
$(\mathtt{A}:x_3)$		
	$(\mathtt{B}:x_4)$	$(\mathtt{C}:x_4)$
		$(\mathtt{C}:x_5)$

Notice that because the positioning of tokens in an Conposit model is insignificant, it turns out that the premises "Some of the X are Y" and "Some of the Y are X" (see next subsection) look essentially the same to Conposit. Equally, the premises "None of the X are Y" and "None of the Y are X" look essentially the same. The reversals may, however, affect the randomly-selected numbers of tokens of the two sets, which in turn affects details of performance to some extent.

Negative Premises

To indicate that a token of a set A is not in set B, it can simply be made *non-optional* and unlinked to any token of class B. Thus, if the first premise is "Some of the artists are not beekeepers," a possible model is:

```
A:x1
A:x2
A:x3      B:x3
          B:x4
```

Here there are some artist tokens that definitely do not represent beekeepers. There is also an artist token that does represent a beekeeper. The premise "None of the artists are beekeepers" leads to a similar sort of model, but no token representing both an artist and a beekeeper is included.

In [25], negative barriers are abandoned, and instead tokens prefixed by a negation symbol are used. In our view, even the use of negation symbols is unnecessary. Johnson-Laird and Byrne's use of them presumably comes from the idea that a token that is not mentioned as *not* being in a given set *might* be in the set nevertheless. Conposit's semantic assumption differs: it takes such a token as definitely not being in the set.

Procedural Adjustments

In [24], Johnson-Laird and Bara hypothesize that in some cases human syllogizers need to switch the order of the premises in order to allow the information from them to be more easily integrated. This re-ordering feature is not included in Conposit, although it would have been conceptually straightforward to include it.

A conclusion produced by a syllogizer is meant to be of the form "C *rel* A" or "A *rel* C". Sometimes the Johnson-Laird theory will first consider the possibility of a conclusion in one of these forms and then, perhaps, consider the other. Processing load problems may, however, cause the second order to be neglected, resulting in the overlooking of a possible conclusion. In the Conposit version, both orders are always tried. However, in line with the fact that Conposit is essentially insensitive to the distinction between premises "Some of the X are Y" and "Some of the Y are X", the system does not bother to produce the conclusion "Some of the C are A" as well as "Some of the A are C." A similar point applies to "None of ..." conclusions.

Given that a conclusion of the form "X *rel* Y" is being sought, the Johnson-Laird theory first tries to produce a *universal* affirmative conclusion (all the X are Y), and only if that fails will it consider a *particular* affirmative one (some of the X are Y). There is a similar pre-empting of particular negative conclusions (some of the X are not Y) in favor of universal negative ones (none of the X are Y). By contrast, Conposit produces the particular versions from a model even when the universal versions can also be produced.

In [24] (pp.38/39), Johnson-Laird and Bara present five diverse rules for modifying the current mental model to get a new one that is still consistent with the premises (but possibly falsifying a current tentative conclusion). Johnson-Laird and Bara (p.37) doubt that human syllogizers search the space of models "either randomly or in a totally systematic way," but their implementation of the theory appears to be largely if not wholly systematic. Instead, Conposit embodies a simple, random model-generation process. When one model has been created and tentative conclusions drawn from it, another one is randomly created, and so on. The possibility that a previous model might be re-generated up to isomorphism, just by chance, is tolerated. This exploration method is much simpler than that in [24], while still letting the modified theory be complex enough to act as a significant challenge to connectionism. Notice that because Conposit considers only randomly many randomly-constructed models, it is possible for more than one totally incorrect conclusion to survive. By contrast, the Johnson-Laird theory never produces more than one conclusion.

Summary of the Modified Processing

The overall syllogistic reasoning process as conducted by Conposit is as follows. Given a pair of premises (expressed in an internal propositional form rather than in natural language), Conposit creates randomly many, randomly-constructed models that are consistent with the premises. Each model is constructed by first creating suitably linked tokens for the sets A and B in the first premise, and then adding in some C tokens, linked to B tokens in a way that is consistent with the second premise. From the first model constructed, as many tentative conclusions as possible are constructed. Subsequent models are used to eliminate tentative conclusions that do not hold in them.

2.3 The Challenge Posed by the Johnson-Laird Theory

The challenge posed by the Johnson-Laird theory has three main parts:

(1) The theory requires at least two sorts of *relatively complex, temporary information structure* to be *rapidly created on the fly*. These are the mental models and tentative syllogism conclusions.

(1a) In particular, mental models involve *variable numbers of co-existing members* of each of several sets.

(1b) Similarly, at any time several temporary propositions *co-exist* in working memory (two premises and possibly a conclusion, at least; and often several tentative conclusions).

(2) The mental model processing is meant to work with *any* three distinct sets in a syllogism (cf. artists, beekeepers and chemists in the above example), and thus raises the issue of *systematicity* of inference [20].

(3) The theory involves *complex procedural control.*

These features are linked to various major, well-known difficulties for connectionism. In particular, the structures in (1) are simple from the point of view of symbolic AI, but complex in the context of connectionism, which has a problem even with the encoding and systematic processing of quite simple structures. The variable binding problem is a particular manifestation of (2).

A full, detailed explanation of exactly why features (1) to (3) impose significant elaboration requirements on connectionism would be involved and lengthy. It would have to take account of the numerous different types of connectionist network, and would have to grapple with the fact that connectionism has no precise, general definition. Nevertheless, detailed arguments that go a long way towards explaining why the features are troublesome are given in [1], [5], [6] and [9]. The following comments only give some brief indications of the nature of the issues.

Feature (1) requires some means for rapidly putting together the components of the information structures, where the combinations formed may be unanticipated in their specifics. For instance, in creating a tentative syllogism conclusion, the system must be able to state any of the four allowed relationships between any two sets A and C whatsoever (cf. feature (2)). Set A might be athletes, set C might be clouds. The system may never before have formed any proposition that relates athletes to clouds. Furthermore, the sets themselves could be novel. Although examples of syllogisms usually use just plural nouns to define the sets, one could have more complex noun phrases such as "the athletes with red hair."

The upshot is that the possible combinations are extremely numerous and arbitrary. This calls into question, for instance, the idea of setting aside one network unit for each possible syllogism conclusion. At the same time, the coexistence in (1b) means that the system cannot simply light up an athletes unit, a clouds unit, and a "some" unit to represent the proposition that some athletes are clouds. Such a scheme would lead to the well-known problem of crosstalk. Other suggestions include facilitating suitable connection paths among the athletes node (unit or sub-assembly), the "some" node, the clouds node, and a recruited node standing for the proposition as a whole. However, this leads to considerable problems in seeing how the constructed proposition could be used in further reasoning, such as determining that the proposition is inconsistent with some mental model. (This is especially so if the facilitation consists of weight changes on the connection paths as opposed to the activation of gating nodes on the paths, but even the latter is troublesome. See especially [1].)

A recent suggestion has been to use reduced representations (see, e.g., [22] [30] [29]) to encode structured data. The difficulty here is (1a) and (1b). Should one overall reduced representation be formed for the entire set of tentative conclusions, model tokens, premises, etc., or should these items be encoded as separate reduced representations sitting in different regions of the network?

The former suggestion is conceptually very tidy, but either requires continual expansion of reduced representations in order to extract individual propositions (etc.) and components of them, or requires a high, and as yet unproven, degree of ability to get the same effect by manipulating reduced representations as wholes. A certain interesting degree of ability on this has been achieved (e.g., [14] [30] [16] [13] [33]), but there is a long way to go before it has been sufficiently extended and elaborated. (Some particular obstacles are detailed in [5] [6].)

On the other hand, if separate reduced representations are maintained, there is an elaborate "storage management" problem. Available regions in which to place new propositions, tokens, etc. need to be found; and either the requisite inferencing circuitry must be replicated across the different regions, or the system must move representations to canonical regions where they can be subjected to inference.

As for (3), the overall process required by the Johnson-Laird theory is much more complex than the overall processes effected by connectionist systems heretofore. Many steps are involved, and there is important iteration and branching. Some of the iteration has unpredictable extent — mental models (in the modified theory) vary in size randomly, but the system must be able to inspect all the tokens in various ways.

Any methods proposed for coping with the challenges should be extensible beyond the Johnson-Laird theory, which is after all only being used as a case study. But features (1) to (3) are just special, highly restricted cases of the general point that high-level cognition involves multiple, co-existing, complex, short-lived, rapidly-created, diverse, novel working memory items, and complex manipulation profiles. For instance, the arbitrariness and novelty of combination in syllogism conclusions is just a pale reflection of the fact that natural language understanding, commonsense reasoning, and so on naturally bring in very arbitrary combinations of concepts. This is especially so when, for instance, mistaken beliefs, metaphor, counterfactual propositions, dreams or children's fiction are at issue.

3 Mental Models in Conposit

This section gives a general overview of Conposit, details the way syllogisms and mental models are realized in its working memory, and briefly sketches the rules that manipulate the contents of working memory. Since this paper is focused mainly on RPE (relative-position encoding) and PSA (pattern-similarity association), it omits a detailed description of the nature of rules and of how they cooperate to work a syllogism. This detail can be found in [2], which also has an appendix detailing simulation results. A portion of those results is included as section 6 below.

The descriptions in the present section are cast at a level of description somewhat higher than that of connectionist circuitry. The connectionist realization is sketched in section 4.

3.1 Overall Structure

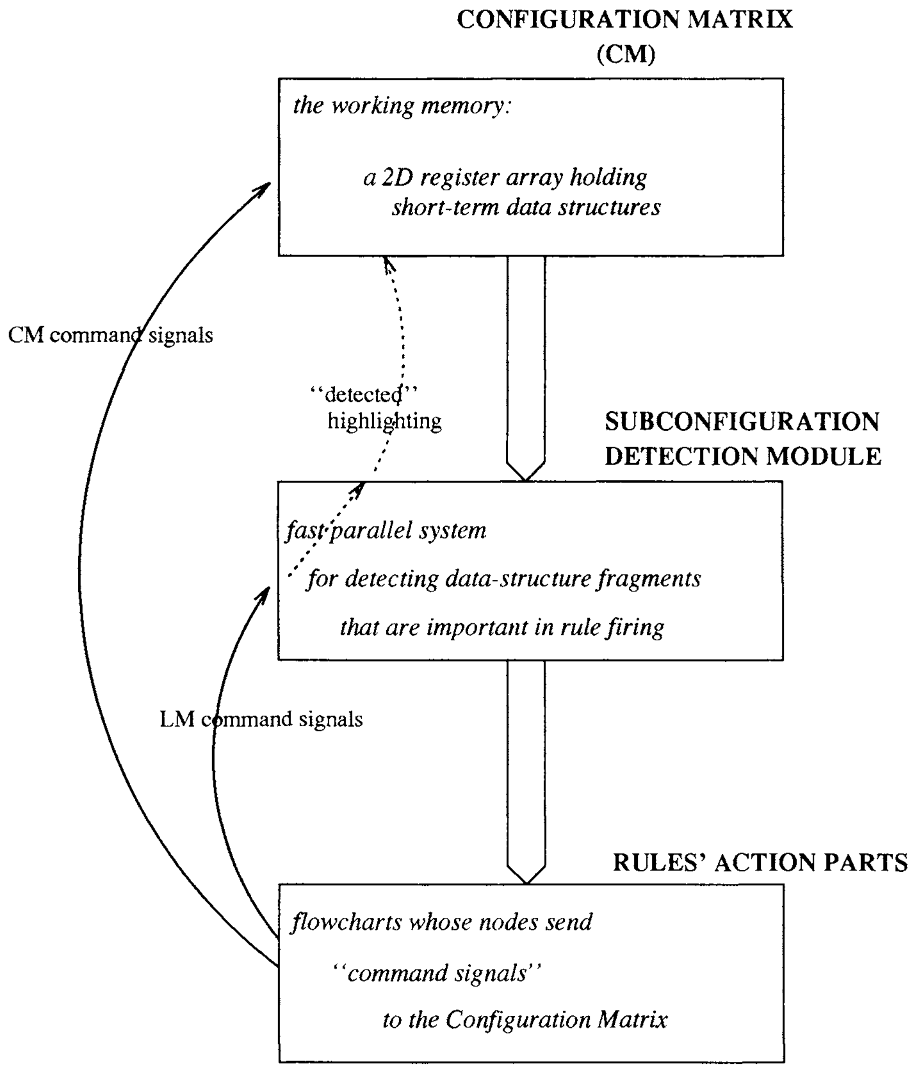

CONFIGURATION MATRIX (CM)
the working memory:
a 2D register array holding short-term data structures
CM command signals
"detected" highlighting
SUBCONFIGURATION DETECTION MODULE
fast parallel system
for detecting data-structure fragments
that are important in rule firing
LM command signals
RULES' ACTION PARTS
flowcharts whose nodes send
"command signals"
to the Configuration Matrix

Figure 1 Overall architecture of Conposit.

The gross architecture of Conposit (and all other previous Conposit versions) is pictured in Figure 1. The system is more or less a conventional rule-based system. The "Configuration Matrix" (CM) is the working memory, and is where syllogism propositions and tokens sit. The action parts of the rules are realized in the Action Parts module. The condition parts of the rules are realized in the Subconfiguration Detection Module, which detects particular data structures in the CM. On each cycle of operation of the system, one action part is chosen randomly out of those that are enabled by the output of the Subconfiguration Detection Module. This action part is executed, usually with the effect of changing the state of the CM. A new cycle then starts.

Each action part can perform a major amount of processing — the rules are quite coarse-grained. Consequently, there are only about ten rule firings per mental model constructed during a simulation. This count includes the rule firings for constructing the model, creating or checking tentative conclusions, and destroying the model.

The initial state of the CM is set by the experimenter. The simulation program contains a procedure that converts syllogism propositions expressed in a convenient list format into data structures within the CM, but this format and the conversion of it are not part of Conposit proper. The simulation program stops when no rules are enabled.

3.2 Configuration Matrix (CM)

The CM, Conposit's working memory, is a 32×32 array of "(active) registers." The state of a register at any time consists of two main portions: a "symbol" and a "highlighting state". Each register is a connectionist subnetwork. The symbol and highlighting state in the register are activity vectors in certain portions of the subnetwork.

Some symbols are called "constant" symbols, and permanently denote particular entities. Denoted entities can be of any sort whatever, including not only people (e.g. individual artists) and classes of people (e.g. the set of all artists), but also situations (e.g., that of a particular artist loving a particular beekeeper), and classes of situations. Any register can contain any symbol, and a symbol may occur in many different registers at the same time. There is also a "null" symbol, which never has a denotation, and a set of variable-like "unassigned symbols," which pick up temporary denotations in a way to be described.

The highlighting state is a vector of ON/OFF values for a tuple of "highlighting flags. Each register has the same tuple of flags, but of course the values need not be the same. Highlighting flag values serve mainly

- to help define relationships among things denoted by the symbols,
- as markers identifying current loci of attention of a rule that is executing,
- as markers in marker passing processes.

There is no in-principle restriction on how many of the highlighting flags can be ON at once in a given register, but in fact only a small number ever are. If a given flag is ON, the register is said to be highlighted in or with that flag.

A register not involved in a representation is said to be *free*. It has a "null" symbol (all symbol-vector units OFF). It also has OFF values on all highlighting flags except some that are used in certain housekeeping activities.

Each representation in the CM consists of one or more separate "clumps" of registers in suitable symbol/highlighting states. A clump consists of a non-free "head register" together with one or more non-free "role registers" immediately adjacent to it. An idea of the nature of a clump can be obtained from Figure 2, which shows only an arbitrary 8×8 region of the CM. The head in this case is the register that is highlighted by a flag called *instance*, which is indicated by the ∇ sign in the figure. There are also three role registers. These are the ones shown with a black dot, "arg1" and "arg2." These three marks indicate certain highlighting flags. The clump encodes the proposition that "all the artists are beekeepers." The ARTISTS and BEEKEEPERS symbols denote the sets of all conceivable artists and beekeepers respectively. The SUBSET symbol denotes the class of all conceivable situations in which one set is a subset of another. Further explanation of the figure will be given in a moment.

Each syllogism premise, tentative conclusion, and token in a mental model is realized as a single clump. Different clumps are separated by free space: that is, no register of one clump is adjacent to a register of another. The absolute positions of the clumps in the CM are irrelevant: what is important is the particular symbols and highlighting states within each clump, and (as will become apparent) the way that clumps *share symbols*.

Also, the only important between-register relationships within a clump are their *adjacency* relationships and certain combinations of highlighting states

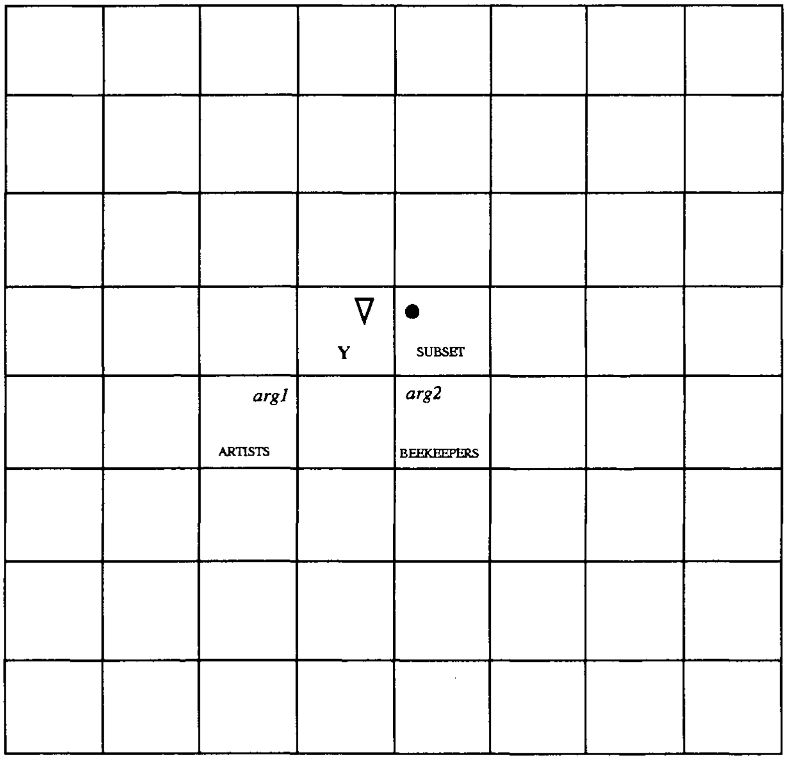


Figure 2 8 × 8 region of the Configuration Matrix, containing a possible clump representing "All the artists are beekeepers."

in adjacent registers. The direction of adjacency is unimportant. For instance, if a clump contains just two registers, they can be "horizontally," "vertically," or "diagonally" arranged in the Configuration Matrix. The arrangement makes no difference to the way the clump is treated by rules.

Representations in the CM are modified when individual registers respond to the "command signals" that rules send to the CM as a whole. A register's response consists of a replacement of its symbol and/or changes to its highlighting flag values. As a result, clumps can rapidly be modified and made to appear and disappear.

3.3 SYLLOGISM PROPOSITIONS IN CONPOSIT

In Figure 2, the register containing the SUBSET symbol is highlighted with a flag called *class*, as indicated by the • sign.[2] The significance of *instance* and *class* highlighting is as follows:

Semantic Stipulation on Class Membership

If a register is currently highlighted with *class* and contains a symbol s denoting a class, and is adjacent to a register that is currently highlighted in *instance*, then the latter register currently denotes some (putative) member of the class denoted by s.

Hence, the *instance*-highlighted register in Figure 2 denotes some subset situation.

Notice that both symbols and registers are regarded as being able to denote. In fact, the following principle applies:

Stipulation on Denotation by Registers and Symbols

If a register currently contains a non-null symbol, then the register and the symbol currently denote the same thing.

The three symbols mentioned so far have a fixed, permanent denotation, and this is borrowed by any register such a symbol temporarily lies in. As will be seen in a moment, the borrowing can go in the other direction, in the case of unassigned symbols. In either case, however, denotation by registers is always temporary, since the presence of a symbol in a register is only ever temporary.

There is a further semantic stipulation to the effect that if a register denotes a subset situation and is highlighted with *instance*, then any neighbor highlighted with a flag called *arg1* denotes the subset, and any *arg2* neighbor[3] denotes the superset. Highlighting with *arg1* and *arg2* is shown in Figure 2.

Set intersection relationships, such as in "some of the beekeepers are chemists," are encoded in a similar way, only using an INTERSECTION symbol instead

[2]The flag names *instance* and *class* replace the names *white* and *black*, respectively, used in other papers on Conposit.

[3]"*arg2* neighbor" means the neighbor highlighted with *arg2*.

of SUBSET. For the proposition just quoted, the *arg1* register would contain the BEEKEEPERS symbol and the *arg2* register would contain the CHEMISTS symbol.

The method described here for encoding propositions is exactly the same as is used in other Conposit versions having nothing to do with syllogistic reasoning. For instance, in another Conposit version the proposition that John loves Mary would be encoded by means of a clump like that in Figure 2, only using a LOVES symbol instead of SUBSET, and JOHN and MARY symbols instead of ARTISTS and BEEKEEPERS. Here the LOVE symbol would denote the class of all conceivable loving situations.

A negative premise/conclusion is realized as pictured in Figure 3. The two clumps together encode the proposition that the ARTISTS set is not a subset of BEEKEEPERS. This involves two clumps linked by virtue of the fact that they each include a register containing the symbol Y. Symbols Y and Z are two of Conposit/SYLL's fifty "unassigned symbols," which have no assigned denotations. Unassigned symbols are regarded as picking up *temporary* denotations from the registers they are in. This is by virtue of the Stipulation on Denotation by Registers and Symbols. Consider the head register containing Y in Figure 3. That register temporarily denotes the situation S of the ARTISTS set being a subset of the BEEKEEPERS set. This makes Y denote that situation as well. Hence, by that same Stipulation, the register containing Y in the other clump also denotes situation S. Hence, this clump represents the situation of S *not* holding. (The assigning of denotations to unassigned symbols and registers is entirely in our minds as observers — it is not a process performed by Conposit itself. Conposit merely processes register states in a way which is consonant with the theoretical assignation of denotations.)

Symbol-sharing is viewed as establishing an association between the registers that share a symbol. Since a symbol is implemented as a connectionist activation pattern, between-clump association by symbol sharing is a simple special case of PSA. On the other hand, the association of registers within a clump is by a simple form of RPE. Within-clump association depends only on the adjacency of suitably-highlighted registers. (See [9] for a description of PSA and RPE in general terms, possible instances of them diverging from Conposit's, and relationship the two techniques bear to other connectionist work and to computer data structuring techniques.)

Propositional embedding, as implemented through PSA, is also used in Conposit to order the components of a syllogism, as illustrated in Figure 4. This shows one possible encoding of the following syllogism:

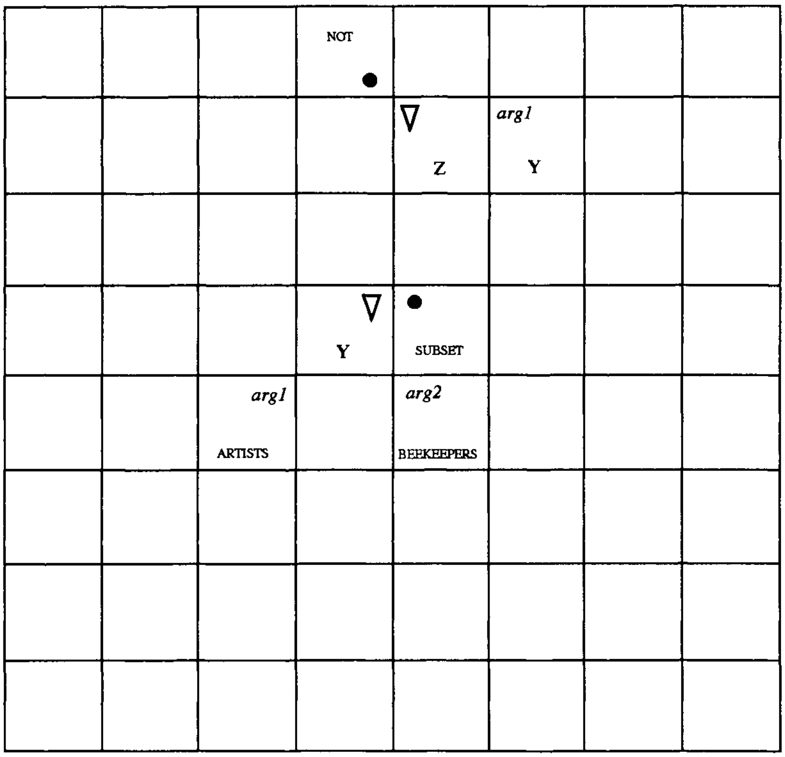


Figure 3 Clumps for "Some of the artists are not beekeepers" (i.e. it is not the case that all artists are beekeepers).

(P1) `All the artists are beekeepers.`

(P2) `Some of the beekeepers are chemists.`

(I) `The relationship of artists to chemists is to be investigated.`

The inclusion of this third item is a respect in which Conposit might be said to be cheating, in that human subjects in syllogism experiments are not always given an explicit command about what sets to relate. However, in [24] Johnson-Laird and Bara do not give an account in their theory of how human subjects work out what sets to relate, and in any case it would be possible to modify Conposit so that it works this out.

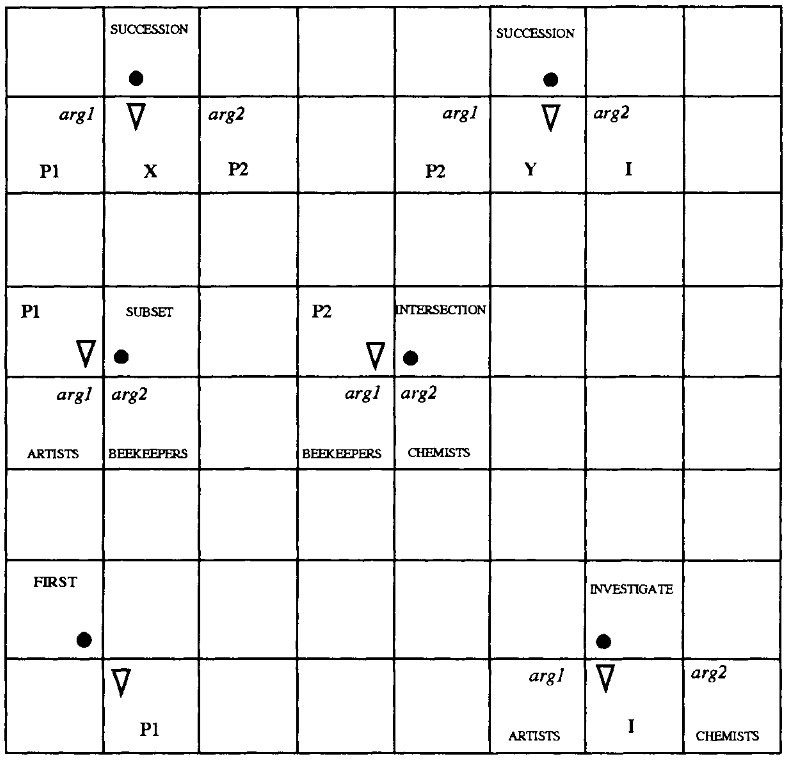


Figure 4 A possible encoding of a syllogism problem.

The proposition (P1) that all the artists are beekeepers is ordered before the proposition (P2) that some of the beekeepers are chemists. The left-hand SUCCESSION clump in the Figure states that whatever is represented by unassigned symbol P2 is to be considered after whatever is represented by P1. P1 and P2 are made to denote the mentioned propositions by virtue of their appearance in the heads of the clumps in the middle of the Figure. The "FIRST" proposition in the Figure states that proposition (P1) is the first item in the sequence of three propositions. A run of Conposit is initialized with a CM state of the sort shown.

In some applications of Conposit there might be propositions with more than seven argument places, which is a problem in view of the fact that a head register in a clump has at most eight neighbors. In order to solve this problem,

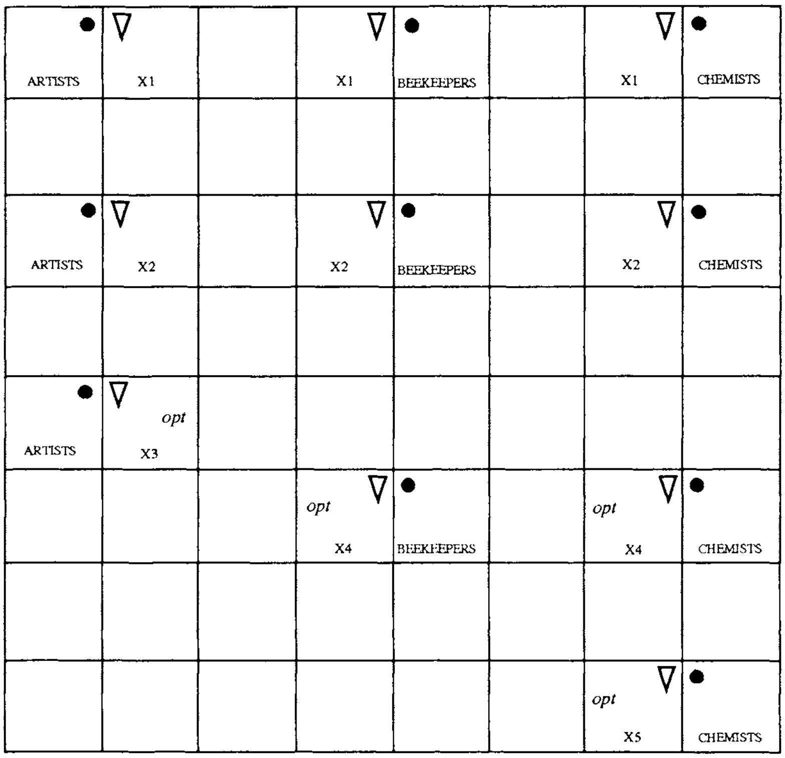


Figure 5 Clumps for mental model MM 2-2.

Conposit allows a proposition to be separated into several clumps encoding different aspects of the proposition [2].

3.4 REPRESENTATION OF MENTAL MODELS

An individual token is encoded by a straightforward application of the above techniques. For instance, the encoding of an artist token consists of a *instance/class* clump, where the *class* register contains the ARTISTS symbol, and the *instance* register contains some unassigned symbol. See the three clumps at the left of Figure 5. The *instance* register in each of these three clumps denotes some artist. So the unassigned symbol in the register also temporarily denotes that artist. In terms of the adjustments to the Johnson-Laird theory that were described in section 2.3, the unassigned symbol acts as the name of the token. The clump corresponds directly to the notation "A:x" used there, if A stands for the artists set. A token is marked as being optional by virtue of its head register being highlighted with a special flag called *optional* (cf. *opt* in bottom left clump of Figure 5.)

Because mental models are randomly constructed one at a time by Conposit, the CM never contains more than one model. The Johnson-Laird mental model MM 2-1, which has the adjusted form MM 2-2, could in principle take the form pictured in Figure 5. However, this arrangement is regimented for illustrative clarity. The positioning of the clumps is, as always, irrelevant, and when they are created they are placed in arbitrary positions in the CM. They can even be arbitrarily placed with respect to the clumps that encode the syllogism itself (Figure 5). That is, the CM is *not* divided into separate regions for propositions and tokens.

3.5 CONPOSIT'S RULES

Conposit has the following rules.

Note-First

Designates the first premise as the "current" proposition by imposing a certain highlighting state on it.

Note-Next

Transfers the "current" designation from first premise to second, when the former has been marked as having been processed. Similarly, the rule transfers the "current" designation from the second premise to the "to be investigated" proposition.

Create-Unclassified-Tokens

If there are no unclassified tokens (as happens if and only if the CM contains no mental model), and some syllogism proposition is "current," this rule creates randomly many "unclassified" tokens. These are just like, say, ARTISTS tokens but using the symbol THINGS rather than ARTISTS.

When one of the rules below creates a proper token, say an ARTISTS one, the token is a modified copy of an arbitrarily chosen unclassified token. The modification consists of replacing the THINGS symbol by the ARTISTS symbol.

Intersection

When an intersection proposition ("Some of the X are Y") is "current" *and* there are some tokens, this rule proceeds as follows.

Unless there are already some X tokens *and* some Y tokens, the rule assumes the proposition is a premise and modifies the model appropriately. The modifications consist mainly of additions of tokens of X and/or Y (unless some X, Y tokens, respectively, already exist), and ensuring that at least one X token has the same name as some Y token.

On the other hand, if there are both some X tokens and some Y tokens, the rule assumes the proposition is a tentative conclusion. In this case it deletes the proposition if and only if it fails to hold in the model.

Subset, Not-Intersection, Not-Subset

These three rules are analogous to *Intersection*, and deal with premises or tentative conclusions of the form "All the X are Y," "None of the X are Y" and "Some of the X are not Y" respectively.

Create-Conclusions

When the *first* mental model has just been created from the premises and therefore the "to be investigated" proposition is "current," this rule creates tentative conclusions about the end terms (the A and C sets) in the syllogism. A tentative conclusion is encoded in just the same way as a premise except that it has its head register highlighted with the *tentative* flag.

The marker passing process that organizes the conclusion creation is detailed in section 3.6 below.

Check-Conclusions

When a mental model *other than the first* has just been created, and therefore once again the "to be investigated" proposition is "current," this rule transfers the "current" designation to the existing set of tentative conclusions, left over from the previous models. This prepares the way for rules *Intersection, Subset, Not-Intersection* or *Not-Subset* to check those tentative conclusions against the current model.

Destroy-Tokens

When tentative conclusions have just been created or checked, this rule destroys the current mental model. The destruction consists simply of the elimination of all tokens, returning each register involved in the clumps to the "free" state.

Remodel

When the current model has just been destroyed and there are some tentative conclusions left, the firing of this rule makes a certain highlighting change that causes *Note-First* to become enabled again. Thus, a new model will be created.

Finish

When the current model has just been destroyed and there are some tentative conclusions left, the firing of this rule deletes all *tentative* highlighting, thereby making the surviving conclusions into definite propositions. This also causes it to be the case that no rules are now enabled. The simulation therefore halts.

Mostly, only one of the above rules is enabled in any given CM state. There are two exceptions to this. First, rules *Remodel* and *Finish* have exactly the same enabling condition, and are therefore always enabled together. The system randomly chooses which one to fire. This is tantamount to the system randomly choosing to try out a new model or to stop. The probability of choosing one of the rules as opposed to the other depends on Conposit parameter settings. The second exception is that when conclusion checking is in progress, more than one of the rules *Intersection, Subset, Not-Intersection* and *Not-Subset* can be enabled. For instance, if there is a tentative intersection conclusion and a tentative negated subset proposition, then rules *Intersection* and *Not-Subset* are both enabled.

If there are two tentative subset conclusions, say, rule *Subset* makes an arbitrary choice of one conclusion to work on. When the rule finishes it removes the "current" designation from this conclusion, so that another firing of the rule will deal with a different subset conclusion.

Much more detail on how the rules work can be found in [2].

3.6 TENTATIVE CONCLUSION CREATION BY MARKER PASSING

Rule *Create-Conclusions* organizes its job by means of a marker passing process that is described here. The presence of a marker at a token consists simply of the highlighting of the head register of the token with one of the flags *member-of-1, member-of-2, member-of-only-one, member-of-both.* The "sharing of token names" mentioned at the start of section 2.2.1 is simply sharing of unassigned symbols by the head registers of tokens. (See Figure 5.)

The process has the following steps. The A and C sets are the two sets involved in the syllogism conclusion.

Initialization

(a) Mark all A tokens with *member-of-1* and all C tokens with *member-of-2.* Spread these marks according to name-sharing (i.e. "identity links"). That is, if a token is marked with *member-of-1* then also mark with *member-of-1* any token with the same name; and proceed similarly with *member-of-2.*

(b) Put *member-of-both* marking on each token that is marked with both *member-of-1* and *member-of-2.*

Symmetric Conclusion Formation

(a) If there are some tokens marked with *member-of-both,* construct the tentative conclusion "Some of the A are C".

(b) If there are no such tokens, construct the tentative conclusion "None of the A are C".

A-C Asymmetric Conclusion Formation

(a) Put *member-of-only-one* marking on all tokens that are marked with *member-of-1* but not *member-of-both.*

(b) If there are some **non-optional** tokens marked with *member-of-only-one*, construct the tentative conclusion "Some of the A are not C".

(c) If there are no tokens marked with *member-of-only-one*, construct the tentative conclusion "All the A are C".

C-A Asymmetric Conclusion Formation

Prepare by deleting *member-of-only-one* marking. Proceed as in A-C case but with A and C interchanged.

3.7 Subconfiguration Detection Module and Rule Enablement

The condition part of a rule consists essentially of a portion of the circuitry in the Subconfiguration Detection Module. This module undeniably contributes the lion's share of the circuitry in Conposit.[4] It consists of a group of interconnected "location matrices" (LMs). Each location matrix is in charge of detecting a particular sort of state subconfiguration within the CM. For instance, one location matrix detects unclassified tokens (like those in Figure 5 but with symbol THINGS in place of ARTISTS etc.). Another detects SUCCESSION propositions that are "ready." A "ready" SUCCESSION proposition is one whose *arg1* register is highlighted in *done*. This highlighting indicates that the proposition denoted by that register has been dealt with, and that it is time to move on to the proposition denoted by the *arg2* register. A further location matrix detects current INTERSECTION propositions. An INTERSECTION proposition is current if its head register is highlighted with *current*.

Notice carefully that there is only *one* LM that detects current INTERSECTION propositions. This LM is insensitive to which sets (ARTISTS, etc.) are involved. Similar comments apply to other LMs. Thus, there is no explosive replication of circuitry within the Subconfiguration Detection Module in order to cope with the possible trios of sets in syllogisms.

Each location matrix is a 2D matrix of the same size as the CM, and the elements are again called registers. For present purposes the reader may take an LM register to have just a binary state, ON or OFF. Suppose the location matrix is the one for detecting current INTERSECTION propositions. Then

[4] It can be dispensed with entirely, however, at the cost of considerably slower processing.

any register in it is ON if and only if the *positionally corresponding* register in the CM is

either the head register of a current INTERSECTION proposition

or a register that contains the same symbol as such a head register.

ON states in LMs arise out of the interconnections between LMs as well as connections going from the CM to LMs.

Several rules may be enabled by the Subconfiguration Detection Module in a given state of the CM. The choice of rule to fire is random, as governed by probabilities provided by Conposit parameters. The enablement condition for a rule is a logical combination of elementary conditions. Each elementary condition is in terms of (a) the presence/absence of ON states in specific LMs, and/or (b) the presence/absence of specific highlighting in the CM. For instance, the enablement condition of the rule *Note-Next* can be paraphrased as:

> "the ready-SUCCESSION LM contains ON somewhere *and* there is no *current* highlighting anywhere in the CM."

The "somewhere" and "anywhere" in the two elementary conditions conjoined here illustrate the point that enablement never depends on *where* a specific highlighting state occurs in the CM or *where* the ONness is in an LM.

3.8 COMMAND SIGNALS

Firing a rule consists largely of the rule's action part sending a sequence of instructions called "CM command signals" to the CM. Each CM command signal goes without modification to *every register in the CM in parallel.* Moreover, the command signal does not contain any sort of name or address for any register, and is therefore unable to explicitly instruct any particular, fixed register to do anything. Nevertheless, different registers may respond differently to the signal, so that in effect the command signal does implicitly cause different registers to do different things. The differences between registers in how they respond to a CM command signal depend mainly on their own current states and the highlighting states of their neighbors. (The CM therefore acts like a grid of processors in an SIMD parallel computer.)

In a simple case, the command signal will dictate that a register is to respond if and only if its highlighting state is as specified by the signal. The command signal may require specific flags already to be ON and/or specific flags already to be OFF, and allows the remaining flags to be at either setting. The register's response then typically consists of changing its highlighting state in the way dictated by the command signal, and/or adopting the symbol conveyed in the command signal. The command signal may also dictate that for a register to be able to respond at all, its *neighbors* must obey a highlighting condition conveyed in the signal. More exactly, the command signal can only require that either *some* or *all* of the register's neighbors have certain highlighting flags ON and/or certain ones OFF.

A command signal often decrees that *arbitrary selection* be done — i.e., that just one, arbitrarily chosen, member of the set of registers that obey the signal's highlighting condition be in the response set. An example of this is when the system makes an arbitrary choice as to which of several different tentative conclusions to check against the mental model. The selection process is performed by the Temporal-Winner-Take-All (TWTA) mechanism, described in section 4.3 below.

The command signal may also decree that *spread by symbol sharing* is to occur (after arbitrary selection if any). This means that any register containing the same symbol as any register already in the response set is now put in the set as well.

A rule action part can also send "LM command signals" that affect the CM indirectly through the medium of LMs. An LM command signal goes to a specific LM, whereupon every ON register in that LM sends a simple signal to the positionally corresponding register in the CM. The CM registers receiving these simple signals then turn on a highlighting flag called *detected*. For instance, if the LM is the current-INTERSECTION one, the CM head register of each current INTERSECTION proposition in the CM turns on its *detected* highlighting. The rule that is firing may now focus its processing on those propositions.[5]

[5] Elsewhere, CM and LM command signals are called "direct command signals" and "indirect command signals" respectively.

3.9 REGISTER RECRUITMENT

Rules often need to create clumps, representing tokens or propositions. A rule creates a clump by means of a short series of CM command signals. The first command signal sets up the required symbol and highlighting values of the clump's head register H, which is arbitrarily chosen out of the set of "ultra-free" registers (see below). It then sets up the symbol and highlighting values for each of the one or more required adjacent registers, choosing each register arbitrarily out of the set of neighbors of the head register. The setting up for each register requires either one or two CM command signals.

When choosing a free register for the head H, the system must ensure that the completed clump will be separated by free space from every existing clump. Let us call a free register "super-free" if it is *entirely surrounded* by free registers. Let us call a *super*-free register "ultra-free" if it is entirely surrounded by *super*-free registers. Register H is chosen arbitrarily out of the set of "ultra-free" registers. The selection of H from among the set of ultra-free registers is done in just the same way that arbitrary selection is done in response to CM command signals, i.e. by another instance of the Temporal-Winner-Take-All mechanism described below. A simple, continuously operating background process in each register maintains two special highlighting flags indicating respectively whether it is super-free or ultra-free. The background processes in different registers operate in parallel with each other.

If the CM is full, i.e. there are no ultra-free registers, then the clump is simply not created. This failure could cause a mental model to be oversimplified by not having some tokens, or could cause a possible tentative conclusion not to appear, but the system would be able to continue its processing. In any case, the CM has not become full in the many simulation runs that have been performed.

4 CONNECTIONIST REALIZATION OF CONPOSIT

This section sketches the quite straightforward way in which Conposit's Configuration and Subconfiguration Detection Module are realized in connectionist circuitry. The most interesting aspect is perhaps the Temporal-Winner-Take-All selection mechanism (section 4.3). For detail on the connectionist realization of rule enablement and action parts, see [2]. An action part is realized as a connectionist subnet structured as a flowchart, most nodes of which send command signals to the CM or LMs.

4.1 CONFIGURATION MATRIX, COMMAND SIGNALS, AND PARALLEL DISTRIBUTOR

Each register in the CM is implemented as a small connectionist subnetwork. Different registers are isomorphic as subnetworks, except for minor variations in the registers at the edges of the CM. The symbol in a register at any given moment is an arbitrary binary activation pattern over some group of units in the register. Also, each highlighting flag is realized as a unit in that register that is either ON or OFF. The symbol and highlighting units maintain their values until changed by a CM or LM command signal.

A CM command signal sent by a rule or routine actually arrives at the CM's "parallel distributor." This module is connected to each register in the CM, and forwards an identical copy of the signal to each register. See Figure 6. The signal is an activation vector, each of whose components is a binary or ternary value. The signal therefore travels from place to place on "cables" consisting of parallel connection paths. The vector is divided up into separate portions, some of which are as follows.

Two portions of the command signal vector specify the symbol (if any) and highlighting that a register must have if it is to respond to the signal. The latter, "own-highlighting," portion has an ON, OFF or DONT-CARE value for each highlighting flag. Another, "neighbor-highlighting" portion of the command signal vector similarly specifies the highlighting that a register's neighbors must have in order for the register to respond to the signal. An immediate effect of the arrival of the command signal at a register is to turn on a "own-condition" unit in the register if and only if it satisfies the own-highlighting condition in the signal. Another effect is to turn on a "neighborly" unit in the register if and only if it satisfies the neighbor-highlighting condition.

Each register is connected uniformly to each of its eight neighbors (or fewer, at the edges of the array). These connections allow a register to sense the highlighting states of its surrounding registers. In more detail, each register R simply receives a connection from the "neighborly" unit of every neighboring register, as pictured in Figure 7. These connections feed both into (a) an OR unit and (b) into an AND unit within R. This scheme allows R to detect (a) whether *some* neighboring register satisfies the neighbor-highlighting condition

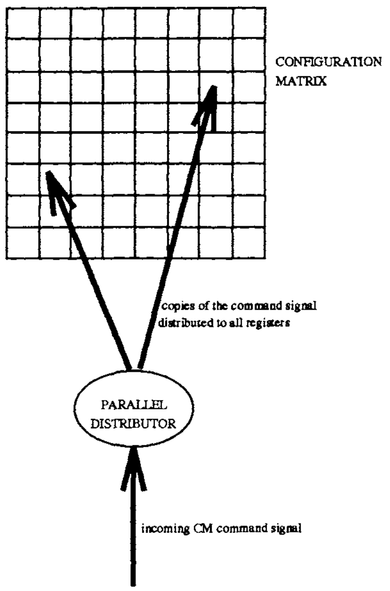

CONFIGURATION MATRIX
copies of the command signal distributed to all registers
PARALLEL DISTRIBUTOR
incoming CM command signal

Figure 6 Parallel distributor, and distribution of CM command signals. The registers can also send certain items of information to the parallel distributor.

in a command signal, and (b) whether *all* neighboring registers satisfy the neighbor-highlighting condition in a command signal. At most one of these decisions is used in determining whether the register should respond to the command signal.

The ability of command signals to make register responses depend on neighbors' highlighting states is part of the means whereby RPE has causal efficacy in the system. The other part is mentioned in the next subsection.

When a certain binary component of a command signal (as an activation vector) is ON, the signal is decreeing that each responding register broadcast its symbol to all other registers.[6] The broadcast is indirect, going via the parallel distributor, as *there is no direct connectivity between non-neighboring registers.* A broadcast symbol is simply held by the parallel distributor until the next command signal arrives, at which time the parallel distributor forwards the symbol to every register. The expectation is that the new command signal

[6] In fact, we assume that in such a case all the responding registers contain the same symbol. In most cases of symbol broadcasting there is only one such register.

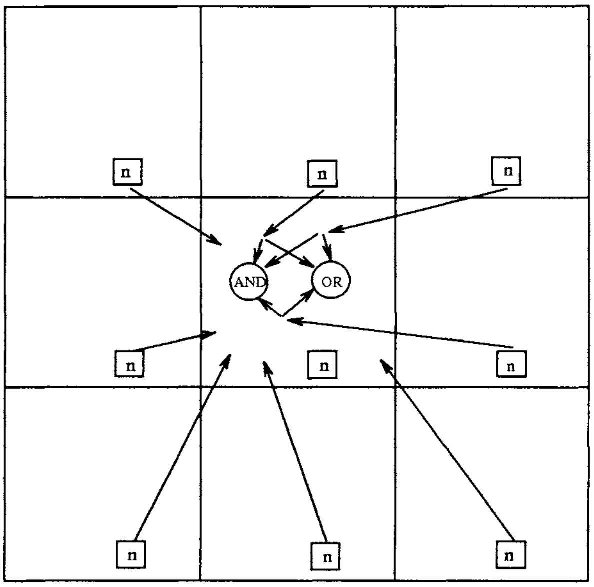


Figure 7 The (only) connectivity between neighboring registers. The large squares depict registers. The small squares containing 'n' depict the "neighborly" units mentioned in the text. (The arrows ending in mid-air depict connections that go to the AND and OR units in the middle register.)

will ordain (by means of an ON value at a certain component of the signal vector) that every responding register take on the broadcast symbol as its new symbol. A symbol travels between registers and parallel distributor on cables of connections.

Broadcasting is also one method that has been used in Conposit to achieve the spread-by-symbol-sharing capability of command signals (see section 3.8). This capability enables the effect of a command signal to be transmitted across the inter-register "linkages" arising from PSA.

4.2 Subconfiguration Detection Module

The connectionist realization of each location matrix in the Subconfiguration Detection Module is analogous to that of the CM. However, it is considerably simpler, since LM registers do not need to decode CM command signals and therefore, in particular, do not need to be connected to neighbors. However, each LM register does need to receive a copy of the symbol in the corresponding CM register. This is to enable ONness to be consistent across positions containing the same symbol, thus respecting PSA "linkages." The mechanism for ensuring this involves symbol broadcasting within the LM. The broadcasts are prevented from conflicting with each other by being arbitrarily sequenced (see below) by a parallel distributor for the LM. This parallel distributor is also involved in the distribution of LM command signals to the LM's registers.

A given LM register is connected to some or all of the following sorts of register in other LMs or in the CM: (i) the positionally corresponding register in the CM; (ii) the positionally corresponding register in another LM; (iii) the registers *neighboring* the corresponding register in the CM; (iv) the registers *neighboring* the corresponding register in another LM. Thus, the LM-LM and CM-LM connectivity is "topographical" to within fan-in from immediate neighbors. The fan-in is important part of the way in which RPE has causal efficacy in the system. More detail of the configuration of the connections can be found in [4].

The LM-LM connections define a directed acyclic network of LMs. Thus, LM states are updated (whenever the CM state changes) in a single forward sweep through the LMs, starting with those connected only to the CM. The set of LMs taking input only from the CM consists of one "basic" LM for every constant symbol and unassigned symbol used by the system. A register in the basic LM for symbol s is ON if and only if the corresponding CM register contains s. This scheme allows a very simple, fast, parallel implementation of the optional "spreading by symbol sharing" phase of a CM register R's response to a CM command signal.

4.3 CM REGISTER SELECTION BY TEMPORAL-WINNER-TAKE-ALL (TWTA)

A CM command signal often decrees arbitrary selection out of some dynamically defined set of contending registers. TWTA is also used within LMs and in rule circuitry, as described in [2].

TWTA is a technique for making a selection out of a set of contending components (units, sub-assemblies) of a connectionist network. It is therefore analogous to conventional connectionist winner-take-all (WTA) mechanisms (see, e.g., [19], [21], [26], [35]; see also [17]). Such a mechanism attempts to select the contending component that is initially most highly active. The selection usually amounts to driving the activations of all the other contending components to some ignorable value (e.g. zero). By contrast, TWTA proceeds on the basis of fine signal timing differences, rather than the activation differences. TWTA is not specific to Conposit - it can be used in any type of connectionist system that allows fine signal timing differences.

TWTA rests on a simple "race" or "first past the post" idea, as follows. Each contending component sends a simple "ready" signal to an arbitration network. The ready signals are generated at times among which there are small random differences, and/or take randomly different amounts of time to travel to the arbitration network. This subnetwork tries to declare as the winner the sender of the ready signal that is earliest to arrive. It does this essentially by using each incoming signal to inhibit the receipt of all the others. However, if several earliest signals are sufficiently close together in time, the arbitration network may not be able to distribute the inhibition fast enough. In such a case, a new round of attempted selection must occur, though now only from among the components whose ready signals scraped through before they could be inhibited. Successive rounds therefore normally use smaller and smaller sets of contenders. On average, the set of contenders is reduced to a fraction f of what it was, where f depends on various network parameters.

By analysis and experiment [11] [12], it has been shown that the expected number of rounds of contention is only roughly logarithmic in the number of initially contending components; and extensive experiments suggest that the actual number rarely exceeds twice the expected number.

In Conposit, an ON value in a certain component of a CM command signal decrees the need for arbitrary selection. A register satisfying the highlighting condition in the command signal sends out its first "ready" signal in an attempt to win the TWTA contention. The arbitration network is part of the parallel distributor. When a register wins, a special "win" highlighting flag within it turns on.

The logarithmic relationship mentioned above, together with the various parameter settings, means that even if all 1024 registers in the CM were initially contending the number of rounds of contention would only be about seven. In fact, in almost all selections, the number of contending registers is far lower. The 1024 figure only arises when the system has to recruit a random register when the CM is entirely empty. After that, further recruitments are done as close as possible to already recruited registers, so that the contending set is much smaller. Also, selection among clumps, such as tokens in a mental model, involves small contention sets.

5 COPING WITH THE JOHNSON-LAIRD CHALLENGE

The present section will show how Conposit meets the requirements posed by the Johnson-Laird theory, through the power provided by its versions of Relative-Position Encoding and Pattern-Similarity Association. A major sub-issue is the question of how Conposit solves the variable binding problem. In [4] and [9] there is more discussion of variable binding in relation to RPE, PSA and other representation techniques.)

Feature (1) of the challenge (see start of section 2.3) is the need to handle multiple sorts of relatively complex, temporary information structure, where these structures are rapidly created on the fly. Conposit's working memory is a pool of recruitable registers. It is easy for the system to recruit however many registers are needed (assuming enough are free) and to structure them into the different sorts of required propositions and token representations. This ease arises from a simple and efficient clump-creation method. All that is required is to put a suitable collection of registers into the right states of activation, in such a way as to achieve the desired RPE and PSA relationships among registers. The CM command signals that achieve this do not need to be sent by rules to any specific registers; rather, command signals are identically broadcast to all registers, and it is the registers' individual states that dictate whether they are to respond to the rule.

The circuitry within the CM that supports all this consist mainly of simple logic within each register, very simple neighbor-neighbor interactions (consisting merely of sensing whether neighbors satisfy a highlighting condition in a CM command signal), simple TWTA arbitration circuitry within the parallel distributor, and trivial circuitry within the parallel distributor for distributing command signals and broadcasting symbols within the CM. The rapidity of data structure creation is ensured in part by having an efficient register selection mechanism (namely TWTA), and by there being an efficient means for each register to determine for itself whether it is free, super-free or ultra-free (section 3.9). These determinations are done in parallel across all registers.

The temporariness of the data structures in the CM is ensured by the fact that non-free registers can easily be put back into the free state, combined with the fact that data structures in the CM have a localist quality, being separated from each other much as symbolic structures in a computer memory are. Therefore, data structures can be individually eliminated without affecting other ones (that are not sub- or super-structures of the ones deleted).

This ease of data structure deletion is brought out especially by the rule that eliminates a whole mental model. All the rule needs to do is to suitably highlight all registers that share an unassigned symbol with a THINGS token, and then eliminate all clumps that are headed by those just-highlighted registers. Each of these actions requires only a couple of command signals. All the token eliminations proceed in parallel with each other.

Feature (1a) in section 2.3 is the need for connectionist systems to be able to manipulate *variable numbers of co-existing members of each of several sets*. In the Johnson-Laird case the members are the tokens and the sets are those mentioned in the syllogism. Conposit's flexibility in this regard is ensured by the fact that the working memory is only a pool of registers that can be used for any representational purpose. No register is permanently dedicated to a specific representational task, such as representing a token of a specific type. Thus, the pool can have any mix of token types that might be necessary at any given time. Several tokens might all be of the same type (all of type ARTIST, say).

Similar comments apply to feature (1b), which is concerned with the co-existence of multiple propositions. Conposit thereby gets away from a narrow view that seems to be adopted in many connectionist discussions of frame instantiation, role-binding, etc. Such discussions often do not take explicit account of the need to be able to have several instantiations of a frame, predicate, etc. co-existing at any one time. The non-dedicatedness of registers is the key to Conposit's handling of this issue.

A further point about the tokens is that they can, when necessary, be manipulated in parallel in Conposit. A simple example of this is the deletion of whole mental models. Another important example is that each step in the marker passing process of section 3.6 is done by manipulating tokens in parallel. Suppose, say, marker *member-of-1* is on several token-head registers R_i, and it is required to copy this marking to any register holding the same (unassigned) symbol as one of the R_i. This effect is achieved by a single CM command signal that does a "spread-by-symbol-sharing" action. What is spread is the ONness of *member-of-1* highlighting. The spreads from different registers R_i occur in parallel, via the medium of the location matrices that correspond to the symbols in the R_i.

Feature (2) in section 2.3 is the need for the mental model processing to work for *any* three classes A, B, C that the system knows about. This is tantamount to saying that the processing rules must have variables that can be bound to whatever classes are involved in the presented syllogism. An idea of how this happens in Conposit can be obtained by looking at the rule *Intersection*, which was briefly described in section 3.5. Neglecting for simplicity the fact that one thing the rule can do is check a tentative conclusion, it can in gross terms be described as follows:

```
IF there is a current INTERSECTION proposition, P,

THEN modify the current mental model according to P.
```

In this formulation there is one variable, P, which gets bound to the intersection premise. The form this binding takes in Conposit is the placing of *detected* highlighting on the head register of the proposition's clump. In effect, variable P is realized as *detected* highlighting as such. This highlighting then allows the rule to access the registers containing the symbols S1 and S2 denoting the sets that are stated to intersect. These registers are just the *arg1*-highlighted and *arg2*-highlighted registers adjacent to the *detected* register, i.e. the head register. What the rule does is to place *set1* highlighting on the *arg1* register, and *set2* highlighting on the *arg2* register. Each of these actions is accomplished very simply by one command signal. For instance, in the first case, this signal just tells all registers to turn their *set1* flags on if their *arg1* flags are on and they are adjacent to a register that has its *detected* flag on. Also, the command signal specifies spread-by-symbol-sharing, so the *set1* highlighting actually actually goes on all registers that contain the S1 symbol. Similarly for S2. *Set1* and *set2* highlighting may now be regarded as being variables denoting the first and second sets involved in the intersection proposition. The rule can perform further actions involving S1 and S2 by restricting command signals to affect only those registers that are highlighted with *set1* or *set2*. (Note that *set1* and *set2* highlighting can be used for other purposes by other rules.)

An important aspect of the variable binding by means of *detected* highlighting is the method by which that highlighting is positioned by the rule. The rule's action part is only executed when there is an ON value *anywhere* within the LM that detects the presence of current intersection propositions. (These are intersection propositions whose head registers are highlighted with *current*.) Thus, the ON values can be viewed as temporarily binding the variable P simultaneously to all the possible current intersection propositions in the CM. The first action of the rule is to send an LM command signal (recall section 3.5) to this LM, causing all the CM registers corresponding to the ON registers in the LM to acquire *detected* highlighting. The next action is a command signal that restricts this highlighting to just one proposition head. The result is that variable P can be regarded as being bound to just one, randomly chosen, current intersection proposition.

It is important to realize that in this discussion the variable P is purely notional. There is no localized circuit component in Conposit that corresponds to it. It is an abstraction from the way ON states are positioned in LMs and of the way particular types of highlighting are positioned in the CM. Of course, the units that support such ON states and highlighting could be regarded as a rather spread-out realization of the variable in the circuitry.

Conposit's variable binding has an interesting "plurality" feature that appears not to be shared by other connectionist frameworks capable of variable binding. Let us say that a highlighting flag that is regarded as a variable (such as the *detected* flag) is a "binding flag." As was explained a moment ago, a binding flag can get turned on at more than one place in the CM. This flagging can be viewed as a "plural" binding.

In some circumstances plural binding allows a useful form of parallelism in a rule's effect. A good example is provided by the deletion of the current mental model by rule *Destroy-Tokens*. In order to do the deletion, all the rule needs to do is to eliminate all the clumps acting as tokens. This it does by putting *detected* highlighting on the head registers of all unclassified tokens, and spreading the highlighting to all registers containing the same symbols. The latter registers are the head registers of all the other, classified, tokens. That is all done by one command signal. The highlighting now has a plural binding to all tokens in the CM. Another command signal suffices to spread the highlighting to the non-head, *class* register in each token as well. Again, that is all parallel. Finally, a further command signal causes all the registers that have *detected* highlighting to be made free, in parallel. Thus, the plural binding of the *detected* highlighting has been used to achieve the in-parallel deletion of all tokens.

Finally, feature (3) in section 2.3 is the need for complex procedural control. This is provided by Conposit in the following way:

- Rules can create data structures and other traces, such as special highlighting states, that affect the activities of later rules. This is illustrated by rule *Note-First* transferring the "current" highlighting state from one proposition to another (see section 3.5).

- The action parts of individual rules have a complex, flowchart-like form. For instance, rule *Create-Conclusions* acts according to the marker-passing process detailed in section 3.6. This process involves several conditional tests, implying a branching structure in the flowchart. Looping is used in the part of rule *Intersection* that creates new tokens in the mental model.
- Branching and looping within action parts can be sensitive to the state of the CM. The direction the processing takes at a branch point (which may be a loop boundary) can be influenced by the presence of a specific highlighting state anywhere in the CM or by specific LMs contain ON registers (anywhere). For instance, the Symmetric Conclusion Formation phase of the marker-passing process mentioned above tests whether any CM register is marked with highlighting flag *member-of-both*.

6 SIMULATION RUNS

This section summarizes the mental model processing performed by two particular runs of Conposit/SYLL, both on the same syllogism. For clarity of illustration, mental models are displayed using identity-link notation, and token names are suppressed. That is, the notation is as in MM 2-1 in Section 2.

In each run the *Finish* rule was disabled, in order better to show how Conposit/SYLL can check and eliminate tentative conclusions. In each run the probabilities of loop exits were set so as to achieve the following expected numbers:

expected number of unclassified tokens created for each model:	8
expected number of tokens put into a set that is so far empty:	3
expected number of tokens added to a non-empty set:	2
expected number of tokens randomly chosen from a non-empty set:	1.5

Of course, the actual numbers involved on individual occasions sometimes depart greatly from these expected values. One extreme effect is for very few unclassified tokens, perhaps just one, to be created. In some cases this can cause Conposit/SYLL to be unable to form a model consistent with the premises. In these situations the model construction is aborted; the existing tokens are destroyed and a new attempt at model construction is made.

Another extreme effect is for so many tokens to be created that the CM becomes full. This can cause failures to generate tentative conclusions, if the model in question is the first, or to eliminate conclusions otherwise. However, this extreme is much less likely than the too-few-tokens extreme, and rarely occurs.

A.1: RUN 1

Premises

```
    Some of the beekeepers are artists.
    All the chemists are beekeepers.
```

Valid Conclusions: None.

Summary of Behavior

Conposit/SYLL produced two affirmative conclusions on the basis of the first model, eliminated one of them because of the second model, and eliminated the remaining one because of the third. The system therefore achieved the correct answer by creating three models.

First Mental Model, A.1-1 (3 unclassified tokens created)

```
    a     =     b     =     c
                (b)
    (a)
```

Tentative Conclusions Generated from First Model

```
    Some of the artists are chemists.
    All the chemists are artists.
```

Mental Model A.1-2 (5 unclassified tokens created)

```
a     =     b     =     c
            b     =     c
(a)
```

Conclusions Eliminated:

```
All the chemists are artists.
```

Mental Model A.1-3 (2 unclassified tokens created)

```
a     =     b
            b     =     c
```

Conclusions Eliminated:

```
Some of the artists are chemists.
```

No conclusions remaining: simulation stops.

A.2: RUN 2

Premises (same as in RUN 1)

```
Some of the beekeepers are artists.
All the chemists are beekeepers.
```

Valid Conclusions: None.

Summary of Behavior

Conposit/SYLL produced three tentative conclusions, two affirmative and one negative, on the basis of the first model. It eliminated the two affirmative ones because of the second model, and eliminated the negative one because of the third. The system again achieved the correct answer by creating three models.

First Mental Model, A.2-1 (20 unclassified tokens created)

```
a     =     b     =     c
a     =     b
(a)
```

Tentative Conclusions Generated from First Model

```
Some of the artists are chemists.
Some of the artists are not chemists.
All the chemists are artists.
```

Mental Model A.2-2 (5 unclassified tokens created)

```
a     =     b
            b     =     c
            b     =     c
            (b)
            (b)
```

Conclusions Eliminated:

```
Some of the artists are chemists.
All the chemists are artists.
```

Mental Model A.2-3 (15 unclassified tokens created)

```
a   =   b   =   c
        b   =   c
        b   =   c
(a)
(a)
(a)
(a)
```

Conclusions Eliminated:

```
Some of the artists are not chemists.
```

No conclusions remaining: simulation stops.

7 DISCUSSION

Conposit is a little difficult to locate on the localist/distributed spectrum. It is certainly not distributed in any usual sense, but neither is it straightforwardly localist. No register is permanently dedicated to representing any particular thing. However, at any one time, a non-free register represents one specific thing. In this way, Conposit could be said to be "transiently localist."

There is a danger, in considering symbolically oriented connectionist systems such as Conposit or BoltzCONS, of thinking that that they are *merely* implementational — that is, of thinking that the connectionist, implementational level has no influence on the symbolic, implemented level. However, Conposit serves as a demonstration of the implementational level affecting the nature of the implemented level. Such bottom-up, implementing-to-implemented effects have been pointed out by other connectionists, for instance by [37] in noting that his implementation of trees in BoltzCONS allows makes it natural to do traversal without a separate stack, whereas such traversal requires special measures in a standard computer implementation. Conposit gets the same effect with tree traversal, because PSA and RPE associations can be traversed in either direction. However, there some other bottom-up effects as well.

The representational facilities of Conposit lead naturally to the idea of establishing identities between tokens by means of name sharing (i.e. sharing of unassigned symbols), rather than by separate linking elements as in Johnson-Laird's own theory and implementation. This is significant change to the conceptual level of the theory. Separate linking elements could have been implemented in Conposit - it would just have been more cumbersome. Equally, when Johnson-Laird's theory is implemented in a standard programming language such as Lisp it is more natural to use separate linking elements (e.g., by putting tokens into lists) than to use name sharing.

Another significant bottom-up effect in Conposit is the use of "random sequencing." For instance, consider the task of going sequentially through a set of representations, applying some processing to each one individually (as in the investigation of the current set of tentative conclusions in Conposit). It is simpler and more efficient for Conposit to select one representation at random, process it, mark it as processed, then select another one, and so on, than to go through the representations in the order defined by some overarching data structure. (An interesting conjecture is that, for many if not most common-sense cognitive tasks involving a conceptually unordered set of items, random sequencing is all that is necessary.)

By contrast, in a standard symbolic framework a set is usually implemented as a structure in which a linear order can naturally be discerned. (The order may be imposed behind the scenes by the programming language implementation, and be unknown to the programmer.) It is simpler and more efficient just to traverse the set in that order than it is to proceed randomly. The issue here is not the relative efficiency of getting to individual members of the set, but rather the housekeeping that would need to be done to discipline the random selections. For instance, suppose the set elements are the values of an array implemented in a standard way in contiguous memory cells in the computer. Hence, all elements can be accessed equally efficiently. Random sequencing would require that, at any stage, a random selection of an element be done only amongst those elements that have not yet been marked as having been selected. But since the marked elements are in arbitrary positions in the array, a random selection operation would have to *scan* through the unmarked elements, rather than just generating a (pseudo)random number and use it to go directly to a position in the array.

Conposit has some similarity to SIMD computer architectures, although it differs from computer architectures in not containing any pointers or stored instructions. (Command signals are an analogue of computer instructions, but they are generated by fixed subnetworks in the rule networks, rather than being items that can dynamically placed in storage locations.) Also, as further discussed in [9], RPE and PSA are highly reminiscent of basic computer techniques for structuring data in memory. Despite these connections to computers, it would be a mistake to claim that Conposit is not really connectionist. It is merely that Conposit has a level of description that can profitably be cast in terms different from the language of connectionist units and links. However, exactly analogous observations can be made about many more standard types of connectionist system. Many such systems can be completely described as doing mathematical operations such as matrix multiplications. These operations are at a higher level than and have no necessary connection to connectionist units and links. Yet, the existence of this mathematical level of description does not lead one to claim that the systems are not connectionist.

The main deficiency of the Conposit as instanced in this chapter is that there is no provision for learning. However, this is a fault in the overall architecture, not in the basic techniques such as RPE, PSA and TWTA. In fact, some ongoing work on Conposit is looking at the use of these techniques in ABR-Conposit [10] [7], a connectionist implementation of analogy-based and case-based reasoning, with various provisions for learning. A significant modification that has been introduced is to have the unassigned symbols be dynamically constructed out of the symbols and highlighting states in the clumps they lie in. The unassigned symbols thereby become somewhat like the compressed encodings or reduced representations in other connectionist research. The use of these encodings in rapid, parallel mechanisms for matching complex data structures, and in mechanisms for structure-sensitive retrieval of analogues or cases from a long-term database, is under investigation.

8 SUMMARY

Conposit copes well with the challenge to connectionism presented by a complex symbolic cognitive theory developed entirely independently. Conposit gains its advantages largely from have a working memory (namely the "Configuration Matrix") that consists of components that can be dynamically recruited for any representational purpose, and that, to within resource bounds, allows any number of highly temporary complex representations to coexist without interference. The RPE (relative-position encoding) and PSA (pattern-similarity association) methods are at the crux of the ability to rapidly set up complex data structures of any required sort. The Temporal-Winner-Take-All selection mechanism is also a useful contribution to the connectionist arsenal of techniques.

ACKNOWLEDGEMENTS

This work was supported in part by grant AFOSR-88-0215 from the Air Force Office of Scientific Research, grant NAGW-1592 under the Innovative Research Program of the NASA Office of Space Science and Applications, and grant CDA-8914670 under the NSF Institutional Infrastructure Program (Small Scale). I am grateful to T. Eskridge, T.-C. Hong, H. Pfeiffer and K. Srinivas for helpful comments and assistance with simulations and system development. I have also benefited from contacts with R. Inder, P.N. Johnson-Laird and K. Stenning.

REFERENCES

[1] Barnden, J.A. (1984). On short-term information processing in connectionist theories. *Cognition and Brain Theory, 7* (1), pp. 25–59.

[2] Barnden, J.A. (1989). Neural-net implementation of complex symbol-processing in a mental model approach to syllogistic reasoning. In *Procs. 11th Int. Joint Conf. on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

[3] Barnden, J.A. (1990). Syllogistic mental models: exercising some connectionist representation and control methods. *Memoranda in Computer and Cognitive Science*, No. MCCS-90-204, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003.

[4] Barnden, J.A. (1991). Encoding complex symbolic data structures with some unusual connectionist techniques. In J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1*. Norwood, N.J.: Ablex Publishing Corp.

[5] Barnden, J.A. (1992a). Connectionism, generalization and propositional attitudes: a catalogue of challenging issues. In J. Dinsmore (ed), *The Symbolic and Connectionist Paradigms: Closing the Gap.* Hillsdale, N.J.: Lawrence Erlbaum. pp.149–178.

[6] Barnden, J.A. (1992b). Connectionism, structure-sensitivity, and systematicity: refining the task requirements. *Memoranda in Computer and Cognitive Science*, No. MCCS-92-227, Computing Research Laboratory, New Mexico State University, Las Cruces, NM 88003.

[7] Barnden, J.A. (1993a). On using analogy to reconcile connections and symbols. In D.S. Levine & M. Aparicio (Eds), *Neural Networks for Knowledge Representation and Inference*, pp.27–64. Hillsdale, N.J.: Lawrence Erlbaum Associates.

[8] Barnden, J.A. (1993b). Time phases, pointers, rules and embedding. *Behavioral and Brain Sciences, 16* (3), pp.451–452. Invited Commentary (on Shastri and Ajjanagadde's "From Simple Associations to Systematic Reasoning").

[9] Barnden, J.A. & Srinivas, K. (1991). Encoding techniques for complex information structures in connectionist systems. *Connection Science, 3* (3), pp.263–309.

[10] Barnden, J.A. & Srinivas, K. (1992). Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. *Int. J. Man-Machine Studies, 36*, pp.221–246.

[11] Barnden, J.A. & Srinivas, K. (1993). Temporal winner-take-all networks: a time-based mechanism for fast selection in neural networks. *IEEE Trans. Neural Networks, 4* (5), pp.844–853.

[12] Barnden, J.A., Srinivas, K. & Dharmavaratha, D. (1990). Winner-take-all networks: time-based versus activation-based mechanisms for various selection goals. In *Procs. IEEE International Symposium on Circuits and Systems*, New Orleans, May 1990.

[13] Blank, D.S., Meeden, L.A. & Marshall, J.B. (1992). Exploring the symbolic/subsymbolic continuum: a case study of RAAM. In Dinsmore, J. (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap*. Hillsdale, N.J.: Lawrence Erlbaum. pp. 113–148.

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

[15] Charniak, E. & Santos, E. (1987/1991). A connectionist context-free parser which is not context-free, but then it is not really connectionist either. In *Procs. 9th Annual Conference of the Cognitive Science Society*. Hillsdale, N.J.: Lawrence Erlbaum. A revised version appears in J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural*

Computation Theory, Vol. 1. Norwood, N.J.: Ablex Publishing Corp., March 1991.

[16] Chrisman, L. (1991). Learning recursive distributed representations for holistic computation. *Connection Science, 3* (4), pp.354–366.

[17] Chun, H.W., Bookman, L.A. & Afshartous, N. (1987). Network Regions: alternatives to the winner-take-all structure. In *Procs. Tenth Int. Joint Conf. On Artificial Intelligence,* pp.380–387. Los Altos, CA: Morgan Kaufmann.

[18] Dyer, M.G. (1991). Symbolic NeuroEngineering and natural language processing: a multilevel research approach. In J.A. Barnden & J.B. Pollack (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol. 1.* Norwood, N.J.: Ablex Publishing Corp. pp.32–86.

[19] Feldman, J. A. & Ballard, D. H. (1982). Connectionist models and their properties. *Cognitive Science, 6,* pp.205–254.

[20] Fodor, J.A. & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: a critical analysis. In S. Pinker & J. Mehler (Eds.), *Connections and symbols,* Cambridge, Mass.: MIT Press, and Amsterdam: Elsevier. (Reprinted from *Cognition, 28,* 1988.)

[21] Grossberg, S. (1988). Nonlinear neural networks: principles, mechanisms, and architectures. *Neural Networks,* 1, 17–61.

[22] Hinton, G.E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence, 46* (1–2), pp.47–75.

[23] Johnson-Laird, P.N. (1983). *Mental models: towards a cognitive science of language, inference and consciousness.* Cambridge, Mass.: Harvard University Press.

[24] Johnson-Laird, P.N. & Bara, B.G. (1984). Syllogistic inference. *Cognition, 16* (1), 1–61.

[25] Johnson-Laird, P.N. & Byrne, R.M.J. (1991). *Deduction.* Hove, U.K.: Lawrence Erlbaum.

[26] Lippmann, R.R. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine,* 4, 4–22.

[27] Oakhill, J.V. & Johnson-Laird, P.N. (1985). The effects of belief on the spontaneous production of syllogistic conclusions. *The Quarterly J. of Experimental Psych., 37A,* pp.553–569.

[28] Oakhill, J.V., Johnson-Laird, P.N., & Garnham, A. (1989). Believability and syllogistic reasoning. *Cognition, 31* (2), pp.117–140.

[29] Plate, T. (1991). Holographic reduced representations. Tech. Report CRG-TR-91-1, Dept. of Computer Science, University of Toronto, Canada M5S 1A4.

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

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

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

[33] Stolcke, A. & Wu, D. (1992). Tree matching with recursive distributed representations. TR-92-025, Computer Science Division, University of California, Berkeley, CA 94704.

[34] Touretzky, D.S. (1990). BoltzCONS: dynamic symbol structures in a connectionist network. *Artificial Intelligence, 46* (1–2), pp. 5–46.

[35] Yuille, A.L. & Grzywacz. (1989). A winner-take-all mechanism based on presynaptic inhibition feedback. *Neural Computation, 1*, pp.334–347.

3

A Structured Connectionist Approach to Inferencing and Retrieval

TRENT E. LANGE

Artificial Intelligence Laboratory
Computer Science Department
University of California
Los Angeles, California 90024

1 INTRODUCTION

Simple connectionist models have generally been unable to perform natural language understanding or memory retrieval beyond simple stereotypical situations that they have seen before. This is because they have had difficulties representing and applying general knowledge rules that specifically require variables, barring them from performing the high-level inferencing necessary for planning, reasoning, and natural language understanding. This chapter describes ROBIN, a structured (i.e., localist) connectionist model capable of massively-parallel high-level inferencing requiring variable bindings and rule application, and REMIND, a model based on ROBIN that explores the integration of language understanding and episodic memory retrieval in a single spreading-activation mechanism.

One of the most difficult parts of the natural language understanding process is forming a semantic interpretation of the text. A reader must often make multiple inferences to understand the motives of actors and to causally connect actions that are unrelated on the basis of surface semantics alone. The inference process is complicated by the fact that language is often ambiguous in multiple ways. Even worse is that context from further input sometimes forces a reinterpretation of a text's meaning.

A relatively unexplored fact about the language understanding process is that it does not exist in a vacuum. As people read text or hold conversations, they are often *reminded* of analogous stories or episodes in memory. The types of memories that are triggered are influenced by context created by the inferences and disambiguations of the understanding process. A full model of the language understanding and memory retrieval processes must take into account the interaction of the two and how they affect each other.

An example of some of the inferencing and disambiguation problems of the language understanding process can be seen in the sentence:

"John put the pot inside the dishwasher because the police were coming." (***Hiding Pot***).

In this sentence, it first seems that John is trying to clean a cooking-pot. But after reading about the police coming, it seems (to many people) that he was instead trying to hide marijuana from the police. This reinterpretation cannot be made without a complex plan/goal analysis of the input — an analysis requiring the ability to make multiple inferences from general knowledge rules.

After people read stories, they are sometimes reminded of similar episodes. For example, after reading ***Hiding Pot***, a person might be reminded of an analogous story that had been read earlier, such as *"Billy put the Playboy under his bed so his mother wouldn't see it and spank him."* (***Dirty Magazine***). Both stories involve somebody hiding something from an authority figure to avoid punishment. This example shows how crucial the understanding process can be to reminding, since the similarities allowing recall can only be recognized after several necessary inferences.

These examples illustrate several of the problems of language understanding and memory retrieval. First, people must make multiple, dynamic inferences very rapidly to understand texts. The speed with which they do so indicates that they probably use a parallel inferencing process to explore several possible interpretations of a text at once. Secondly, they need to be able to integrate multiple sources of evidence from context to disambiguate words and between interpretations, implying that some sort of constraint satisfaction process is used along with the inferencing process. Equally important is that the interpretations from these language processes affect the kinds of episodes and analogies that people retrieve from memory. The episodes recalled from memory often then affect the interpretation process in turn.

These problems have presented great difficulties for artificial intelligence and connectionist models. Traditional symbolic artificial intelligence models can perform the inferencing necessary for comprehension, but have not scaled up well because of their lack of parallelism and their difficulties with disambiguation. In contrast, connectionist (or neural network) models have parallelism and can perform some disambiguation, but have generally had difficulties performing symbolic processing and inferencing.

This chapter focuses on the development of connectionist models that are able to handle these high-level reasoning tasks in a computationally and psychologically plausible manner. It describes ROBIN (Lange and Dyer [19], Lange [20]) and REMIND (Lange and Wharton [21]), two structured connectionist models that provide a potential explanation for these abilities. Several developments in connectionist variable binding and parallel dynamic inferencing abilities were achieved to create ROBIN (ROle Binding and Inferencing Network), allowing it to perform high-level inferencing and disambiguation difficult for other connectionist and symbolic models of natural language understanding. Using ROBIN as a base, we are also developing REMIND (Retrieval from Episodic Memory by INferencing and Disambiguation), a model that explores the integration of language understanding and episodic memory retrieval with a single spreading-activation mechanism. The integration of episodic memory retrieval with ROBIN's inferencing abilities allows REMIND to explain many psychological memory retrieval results that retrieval-only models cannot.

The current versions of ROBIN and REMIND have had a great deal of success handling the problems of language understanding and memory retrieval. The extensions (in progress) to both models discussed in this chapter will improve both of their respective capabilities and allow us to explore how well their initially promising results scale up. These extensions should prove to be a significant addition to the reasoning and retrieval abilities of connectionist models in general.

1.1 LANGUAGE UNDERSTANDING AND HIGH-LEVEL INFERENCING

The part of the natural language understanding process that this chapter concentrates on is the problem of *high-level inferencing* (Lange and Dyer [19]). High-level inferencing is the use of knowledge and rules about the world to build new beliefs about what is true. To understand a text, a reader must often make multiple inferences to understand the motives of actors and to causally connect actions that are unrelated on the basis of surface semantics alone. Complicating the inference process is the fact that language is often both lexically and conceptually ambiguous. As an example of some of the problems arising in high-level inferencing, contrast ***Hiding Pot*** (*"John put the pot inside the dishwasher because the police were coming"*) with the following example:

"John put the pot inside the dishwasher because company was coming." (***Dinner Party***)

In ***Dinner Party***, most people would infer that John transferred a Cooking-Pot inside a dishwasher to get the Cooking-Pot clean. In ***Hiding Pot***, however, it seemed more likely that John was trying to hide his Marijuana from the police. There is therefore a conflict between the interpretation suggested by the first clause alone (that John was cleaning a cooking-pot) and the final interpretation suggested by the first clause combined with the second clause (that John was hiding marijuana). Even recognizing this conflict subconsciously to allow the potential for reinterpretation clearly requires a number of inferences. For example, ***Hiding Pot*** does not explicitly state that if the police see John's marijuana, then they might arrest him for possessing an illegal object (I1), that John doesn't want this to happen (I2), that he might therefore use a hiding plan to stop them from seeing it (I3), and that being inside an opaque dishwasher might be acting as his plan to stop them from seeing it (I4). All of these inferences are necessary to recognize the most probable causal relationship between the different actions. Shastri and Ajjanagadde [33] have called this kind of processing *reflexive reasoning*, since the inferences must be made very quickly, without readers necessarily even consciously noticing that they have made them.

To understand episodes such as ***Dinner Party*** and ***Hiding Pot***, a system must be able to dynamically make chains of inferences and temporarily maintain them with a variable-binding mechanism. For example, a system must know about the general concept (or frame) of an actor transferring himself to a location (*"coming"*). To initially represent the phrase *"police were coming"* in ***Hiding Pot***, the system must be able to temporarily maintain a particular instantiation of this *Transfer-Self* frame in which the Actor role (a variable) is bound to *Police* and the Loc role is bound to some unknown location (which should later be inferred to be the location of John). The system must also have the general knowledge that when an actor transfers himself to a location, he ends up in the proximity of that location, which can be represented as the rule:

R1: (Actor X *Transfer-Self* Loc Y) *results-in* (Actor X *Proximity-Of Obj* Y)

Applying this rule to the instantiation of the police Transfer-Self would allow the system to infer that the police will be in the proximity of John and his marijuana. Another rule the system must have to understand ***Hiding Pot*** is that a precondition of seeing an object is being in proximity of it:

R2: (Actor X *Proximity-Of* Obj Y) *precond-for* (Actor X *See-Object* Obj Y)

Applying this rule to the new knowledge that the police will be in the proximity of John, the system can infer that there is the potential for the police to see John and his marijuana (I1). The rest of the inferences required to understand ***Hiding Pot*** are the result of the application of similar knowledge rules about the world.

Even the ability to maintain variable bindings and apply general knowledge rules of the above sort is often insufficient for language understanding and other high-level cognitive tasks. This is because the ambiguity of language and multiple possible explanations for actions often leads to several possible interpretations of a text (as illustrated by ***Hiding Pot***). A system must therefore have some means to select *between* those different possible interpretations to choose the most plausible in a given context. One of the fundamental problems in high-level inferencing is thus that of *frame selection* (Lytinen [22]; Lange and Dyer [19]). When should a system make inferences from a given frame instantiation? And when conflicting rules apply to a given frame instantiation, which should be selected? Only a system that can handle these problems will be able to address the following critical subparts of the frame selection problem:

- *Word-Sense Disambiguation*: Choosing the contextually-appropriate meaning of a word. In ***Dinner Party***, the word *"pot"* refers to a Cooking-Pot, but when ***Hiding Pot*** is presented, the evidence is that its interpretation should change to Marijuana.
- *Inferencing:* Applying causal knowledge to understand the results of actions and the motives of actors. There is nothing in ***Hiding Pot*** that explicitly states that the police might see the pot, or even that the police will be in proximity of it and John. Nor is it explicitly stated what the police will do if they see he possesses Marijuana (I1). Each of these assumptions must be inferred from the surface facts of the text.
- *Concept Refinement:* Instantiating a more appropriate specific frame from a general one. In ***Dinner Party***, the fact that the pot was put inside a dishwasher tells us more than the simple knowledge that it was put inside a container. In contrast, the salient point in ***Hiding Pot*** is that it is inside of an opaque object, which allows us to infer that the police will not be able to see it (I4).
- *Plan/Goal Analysis:* Recognizing the plan an actor is using to fulfill his goals. In ***Dinner Party***, John has put the pot into the dishwasher as part of the *Dishwasher-Cleaning* script (a stereotypical sequence of actions)

to satisfy his goal of getting the pot clean, which perhaps itself serves as part of his plan to prepare for company coming over. In ***Hiding Pot***, however, it appears that John has put the pot into the dishwasher to satisfy his sub-goal of hiding the pot from the police (I3), which is part of his overall goal of avoiding arrest (I2).

High-level inferencing is complicated by the effect of additional context, which often causes a *reinterpretation* to competing frames. For example, the interpretation of ***Hiding Pot*** can change again if the next sentence is:

P3: "They were coming over for dinner in half an hour."

P3 provides more evidence for the possibility that John was trying to clean the pot to prepare for dinner, perhaps causing the word pot to be reinterpreted back to a cooking-pot, as in ***Dinner Party***. These examples clearly point out two sub-problems of frame selection, those of frame commitment and reinterpretation. When should a system commit to one interpretation over another? And if it does commit to one interpretation, how does new context cause that interpretation to change?

2 LANGUAGE UNDERSTANDING AND MEMORY RETRIEVAL MODELS

To perform semantic language understanding of inputs for short texts of the type described here, a system must be able to (1) perform the high-level inferencing necessary to create causal plan/goal analyses of the cue, (2) dynamically hold the resulting representations' structure and bindings, and (3) perform lexical and pragmatic disambiguation (and possible reinterpretation) to select the most contextually-appropriate representation. In this section we discuss several related symbolic and connectionist approaches to these language understanding problems and give a brief overview of previous models of episodic memory retrieval.

2.1 SYMBOLIC RULE-BASED SYSTEMS

Symbolic rule-based systems have had the most success performing the high-level inferencing necessary for natural language understanding. A good example is BORIS (Dyer [16]), a program for modeling in-depth understanding of relatively long and complex stories. BORIS has a symbolic knowledge base containing knowledge structures representing various actions, plans, goals, emotional affects, and methods for avoiding planning failures. When a story is read in, BORIS fires rules from its knowledge base to infer additional story information. This allows BORIS to form an elaborated representation of the story, about which it can then answer questions. Other models that have successfully approached complex parts of the language understanding process have all had similar types of knowledge representation and rule-firing capabilities (cf. Schank and Abelson [31]; Wilensky [40]; Lytinen [22]).

While traditional symbolic models have demonstrated an ability to understand relatively complex stories in limited domains, they have encountered difficulty when trying to resolve and reinterpret ambiguous input. One solution has been to use expectation-based conceptual analyzers, such as used in BORIS. These systems use bottom-up or top-down *requests* or *demons* that are activated as words are read in. A word is disambiguated when one of the request rules fires. An example of a bottom-up request that might be used to disambiguate the word pot would be:

If the context involves *Cleaning*, then interpret *"pot"* as a *Cooking-Pot.*

Once such a request is fired, the interpretation chosen is generally used throughout the rest of the inferencing process, and the word is thrown away. However, this makes it impossible to reinterpret the word if the context changes, such as in ***Hiding Pot***. A partial answer might be to keep words around in case a new context causes another disambiguation request to fire. However, this solution creates a different problem — how to decide between conflicting disambiguation rules. For example, one cannot simply specify that the *"pot"* disambiguation request involving the *Police* context always has a higher priority than the request involving the Cleaning context, because police *can* be in the same place as cooking pots (e.g., if ***Hiding Pot*** was followed by *"They were coming over for dinner in half an hour."*) As the amount of knowledge stored in the system grows, the number of disambiguation requests needed grows with them, producing even more conflicts. Moreover, because rule application in traditional symbolic models is fundamentally serial, these systems slow down dramatically as the number of inferencing and disambiguation rules increases.

2.2 MARKER-PASSING NETWORKS

Marker-passing models operate by spreading symbolic markers in parallel across labeled semantic networks in which concepts are represented by individual nodes. Possible interpretations of the input are formed when marker propagation results in a path of units connecting words and concepts from the input text. Like rule-based systems, marker-passing systems are able to perform much of the high-level inferencing necessary for language understanding because of the symbolic information held in their markers and networks (cf. Charniak [5]; Riesbeck and Martin [28]; Granger, Eiselt, and Holbrook [11]; Norvig [26]; Kitano, Tomabechi, and Levin [17]). The primary advantage of marker-passing networks over traditional symbolic, rule-based systems is that their massively parallel marker-passing process allows them to generate all of the different possible interpretations of a text in parallel. This becomes particularly important for large knowledge and rule-bases needed for complex language tasks.

Marker-passing systems have many of the same problems as traditional symbolic systems in performing disambiguation and reinterpretation. Because of the generally all-or-none symbolic nature of the inference paths generated by the marker-passing process, these systems have problems selecting the most contextually-sensible interpretation out of all the paths that they generate. Most marker-passing models attempt to deal with this problem by using a separate symbolic path evaluation mechanism to select the best interpretation. Unfortunately, the marker-passing process generally creates an extremely large number of *spurious* (i.e., unimportant or logically impossible) inference paths, which often represent over 90 percent of the paths generated even for small networks (Charniak [5]). As network size increases to include more world knowledge, there is a corresponding explosion in the number of paths generated. Because path evaluation mechanisms work serially, this substantially diminishes the advantage of generating inference paths in parallel. This explosion of generated connections and the generally all-or-none nature of marker-passing inference paths becomes an especially difficult problem when applying marker-passing systems to ambiguous natural language texts (Lange [20])[1].

[1]Partial solutions to these problems using *hybrid* marker-passing networks that include aspects of spreading-activation have been proposed (cf. Kitano et al. [17]; Hendler [13]).

2.3 Distributed Connectionist Networks

Distributed connectionist (or PDP) models represent knowledge as patterns of activation within massively parallel networks of simple processing elements. Distributed connectionist models have many desirable properties, such as learning rules that allow stochastic category generalization, noise-resistant associative retrieval, and robustness against damage (cf. Rumelhart et al. [29]).

McClelland and Kawamoto's [23] case-role assignment model provides a good example of how distributed connectionist models have been used to model language understanding. The main task of their model is to learn to assign proper semantic case roles for sentences. For example, given the syntactic surface form of the sentence *"The boy broke the window"*, their network is trained to place the semantic microfeature representation of *Boy* in the units representing the Agent role on the output layer. But when given *"The rock broke the window"*, it is trained to place the representation of *Rock* in the Instrument role. Their network is also trained to perform lexical disambiguation, for example, mapping the pattern for the word *"bat"* to a *Baseball-Bat* for sentences such as *"The boy hit the ball with the bat"*, and to a *Flying-Bat* for sentences such as *"The bat flew."* Once the input/output pairs have been learned, the network exhibits a certain amount of generalization by mapping the case roles and performing lexical disambiguation for new inputs that are similar to the training sentences.

One of the main limitations of McClelland and Kawamoto's model for language understanding is that it can only successfully analyze direct, one-step mappings from the input to the output. This limits the model to sentences that can be understood and disambiguated based solely upon the surface semantics of the input. Two distributed connectionist models that get around this limitation are those of Miikkulainen and Dyer [25] and St. John [34]. Both models use *recurrent networks* with a hidden layer of units whose activation pattern essentially stores the state (or "gestalt") of the stories being understood. This allows them to learn to process more complex texts based on stereotypical scripts and script-like stories (Schank and Abelson [31]). Both models have the lexical disambiguation abilities of McClelland and Kawamoto's model, but are also able to infer unmentioned story events and role-fillers from the script that has been recognized by the hidden layer.

Unfortunately, there may be significant problems in scaling distributed connectionist models to handle more complex language. Both the Miikkulainen/Dyer and the St. John model work by resolving constraints from the context of the input to recognize one of their trained scripts and to instantiate it with the bindings of the particular input story. However, much of language understanding involves the inference of causal relationships between events for completely novel stories in which no script or previously trained input/output pair can be recognized. This requires *dynamic inferencing* — producing chains of inferences over simple known rules, with each inference resulting in a potentially novel intermediate state (Touretzky [37]). Most importantly, the problem of ambiguity and the exponential number of potential causal connections between two or more events requires that multiple paths be explored in parallel (the forte of marker-passing networks). It remains to be seen whether a single blended activation pattern across the bank of hidden units in a recurrent network can solve this problem by simultaneously holding and making dynamic inferences for multiple, never-before encountered interpretation chains.

Other distributed models explicitly encode variables and rules, such as the models of Touretzky and Hinton [38] and Dolan and Smolensky [8]). Consequently, such rule-implementing distributed models are able to perform some of the dynamic inferencing necessary for language understanding. However, the types of rules they can currently encode are generally limited. More importantly, like traditional rule-based systems, they are serial at the knowledge level — i.e., they can fire only one rule at a time. As previously mentioned, this is a serious drawback for natural language understanding, particularly for ambiguous text, in which the often large number of multiple alternative inference paths must be explored in parallel (Lange [20]).

2.4 STRUCTURED SPREADING-ACTIVATION MODELS

Structured (or localist) spreading-activation models are connectionist models that represent knowledge in semantic networks like those of marker-passing networks, but in which the nodes are simple numeric units with weighted interconnections. The activation on each conceptual node generally represents the amount of *evidence* available for its concept in a given context. As in marker-passing networks, structured connectionist networks have the potential to pursue multiple candidate interpretations of a story in parallel (i.e. be parallel at the knowledge level) as each interpretation is represented by activation in different local areas of the network. A potential advantage over marker-passing networks, however, is that the evidential nature of structured spreading-activation networks make them ideally suited to perform lexical disambiguation. Disambiguation is achieved automatically as related concepts under consideration provide graded activation evidence and feedback to one another in a form of constraint relaxation (cf. Cottrell and Small [6]; Waltz and Pollack [39]; Kintsch [16]).

Until recently, the applicability of structured connectionist models to natural language understanding has been severely hampered because of their difficulties representing dynamic role-bindings and performing inferencing. The basic problem is that the evidential activation on structured networks' conceptual units gives no clue as to *where* that evidence came from. The networks can therefore tell which concepts are activated, but have no way of determining which roles concepts are dynamically bound to (see discussion in Lange [20]). More importantly, without a mechanism to represent such dynamic bindings, they cannot propagate bindings to make the chains of inferences necessary for understanding more complex texts. Thus, unlike marker-passing systems, most structured connectionist models have been limited to processing simple sentences that can be resolved on the surface semantics of the input alone (e.g., *"The astronomer married the star"*, Waltz and Pollack [39]).

One way of compensating for the lack of dynamic inferencing abilities in spreading-activation networks is to use a symbolic processing mechanism external to the spreading-activation networks themselves to perform the variable binding and inferencing necessary for language understanding. Such a spreading-activation/symbolic hybrid has been used in Kintsch's [16] construction-integration model of language comprehension. This system uses a traditional symbolic production system to build symbolic representations of the alternative interpretations of a text. These representations are then used to construct a spreading-activation network in which the different interpretations compete to integrate contextual constraints. The integration of constraints with spreading-activation in the network allow Kintsch's model to correctly disambiguate and interpret input sentences. A somewhat similar approach is taken by ACT* (Anderson [1]), a psychologically-based spreading-activation model of language understanding, fact encoding and retrieval, and procedure encoding and retrieval. Kintsch's and Anderson's models both illustrate many of the impressive emergent properties of spreading-activation networks for modeling realistic language understanding, such as their ability to model the time course of lexical disambiguation in a way consistent with psychological evidence. However, if a mechanism internal to the networks (instead of an external symbolic production system) could be found to construct text inferences, the parsimony and psychological realism of structured spreading-activation networks would be greatly increased.

Recently, a number of researchers have shown how structured connectionist models can handle some variable binding and inferencing abilities within the networks themselves (e.g., Barnden [2]; Bookman [7]; Holldobler [26]; Shastri and Ajjanagadde [33]; Sun [35]). Most of these models, however, have no mechanisms for handling ambiguity or frame selection. An exception is ROBIN [19], a structured spreading-activation model that propagates *signatures* (activation patterns that identify the concept bound to a role) in order to generate all possible interpretations of an input text in parallel. At the same time, ROBIN uses the network's evidential constraint satisfaction to perform lexical disambiguation and selection of the contextually most plausible interpretation. Thus, ROBIN is able to perform high-level inferencing and disambiguation within the structure of a single network, without the need for external symbolic processing. The following sections describe ROBIN and the extensions we are planning to make to further its inferencing and language understanding abilities.

2.5 MEMORY RETRIEVAL MODELS

The process of memory retrieval has generally been explored in isolation from the process of language understanding. However, remembering and retrieving complex episodes requires many of the same representational, binding, and inferencing abilities that natural language understanding does. Because connectionist models have had difficulties handling complex structural relationships in general, few attempts have been made to build connectionist retrieval models for the type of high-level episodes discussed in this paper.

Nonetheless, a few models have shown the potential value of connectionist models for memory storage and retrieval. For example, COPYCAT (Hofstadter and Mitchell [14]) uses connectionist constraint-satisfaction in solving letter-string analogy problems. Although the retrieval portion of COPYCAT only retrieves simple concepts and not memory episodes, it seems to exhibit some of the fluidity of concepts and perception apparent in human analogical reasoning. DISCERN (Miikkulainen [24]) shows how a variant of distributed connectionist topological feature maps can be used to store and retrieve script-based stories. Besides showing how purely-distributed connectionist models can store and retrieve multiple-sentence episodes, DISCERN exhibits a number of features of human episodic memory, such as certain kinds of memory confusions and recency effects. Although COPYCAT and DISCERN are only able to store and retrieve relatively simple or stereotypical episodes, they do illustrate some of connectionist models' promise for psychologically-plausible memory retrieval.

Symbolic models have had the greatest success in modeling retrieval of complex, high-level memory episodes. Case-based reasoning (CBR) models (cf. Kolodner et al. [18]; Hammond [12]; Riesbeck and Schank [27]; Schank and Leake [32]; Barnden and Srinivas [3]) form the largest class of symbolic memory retrieval models. In CBR models, memory access is performed by recognition of meaningful *index patterns* in the input that allow retrieval of the episodes (or cases) most likely to help them solve their current problem. An analysis phase is usually performed to determine the indices that are most important for finding relevant cases for a particular problem, such as cases that share similar plans, goals, enabling preconditions, or explanation failures. In addition, CBR models are usually careful to retrieve *only* those cases that will help find a solution, explicitly rejecting cases that do not. CBR models are therefore generally models of expert reasoning within a given domain of expertise, rather than models of general human reminding. It is quite possible that expert memory retrieval may be satisfactorily modeled by such methods. However, general reminding seems to be substantially messier, being affected by not only by the sort of useful abstract indices used in CBR models, but also by superficial semantic similarities that often lead to quite *inexpert* remindings. Further, the problem of selecting and recognizing appropriate indices becomes substantially more difficult when reading ambiguous texts outside of limited expert domains.

General, non-expert reminding has been modeled in systems such as ARCS (Thagard et al. [36]) and MAC/FAC (Gentner and Forbus [10]). These systems model retrieval without using specific indexing methods. Instead they retrieve episodes whose representations share superficial semantic similarities with retrieval cues, with varying degrees of preference towards retrieval of episodes that are also analogically similar or structurally consistent. However, unlike most CBR models, these systems do not specify how they construct the representation of retrieval cues from a source input or text, and so cannot explain how inferences and comprehension affect reminding.

REMIND (Lange and Wharton [21]) is a structured spreading-activation model of general non-expert reminding based on ROBIN in which memory retrieval is a side-effect of the language understanding process. REMIND performs dynamic inferencing and disambiguation to infer a conceptual representation of its input cues, as in ROBIN. Because stored episodes are connected to the concepts used to understand them, the spreading-activation process also activates any memory episodes in the network that share features or knowledge structures with the cue. After a cue's conceptual representation is formed, the network recalls the memory episode having the highest activation. Since the inferences made from a cue often include actors' plans and goals only implied in a cue's text, REMIND is able to get abstract, analogical remindings that would not be possible without an integrated understanding and retrieval model.

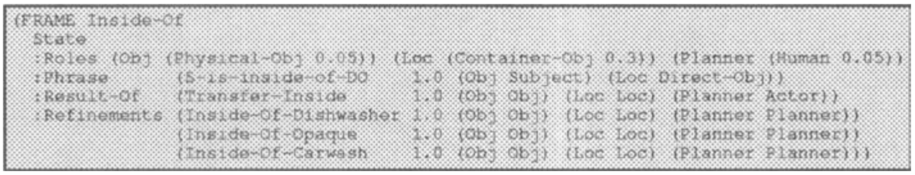
3 INFERENCING IN ROBIN

Our approach is to develop and explore structured connectionist networks that build upon the advantages of spreading-activation networks and that are capable of supporting the processing abilities necessary for language understanding. To this end, we have developed ROBIN, a structured spreading-activation model that is capable of performing dynamic inferencing to generate multiple possible interpretations of an input text in parallel. At the same time, ROBIN uses evidential constraint satisfaction within the network to allow it to automatically disambiguate to the most plausible interpretation in a given context. This section gives an overview of how ROBIN uses these abilities to perform high-level inferencing and disambiguation for natural language. A more detailed description is provided in Lange and Dyer [19] and Lange [20].

3.1 KNOWLEDGE GIVEN TO ROBIN

ROBIN uses structured networks of simple connectionist units to encode semantic networks of frames representing world knowledge. Each frame has one or more roles, with each role having selectional restrictions (i.e. expectations or type restrictions) on its fillers. General knowledge rules used for inferencing are encoded as interconnected pathways between corresponding roles. The knowledge base of frames and rules consists of the causal dependencies relating actions, plans, goals, and scripts (Schank and Abelson [31]) necessary for understanding stories in a limited domain. The knowledge base is hand-built, as in most structured connectionist models. However, there is no information in the knowledge base about specific inputs (such as ***Hiding Pot***, ***Dinner Party***, and ***Dirty Magazine***) that the networks will be used to understand.

Figure 1 gives an example of how knowledge is defined in ROBIN. It defines the conceptual frame *Inside-Of*, which represents the general state of one object being inside of another. *Inside-Of* has three roles: an Object that is inside of something (which must be a *Physical-Obj*), a Location that the Object is inside of (which must be a *Container-Obj*), and a Planner that may have caused the state to be reached (which must be a *Human*). *Physical-Obj*, *Container-Obj*, and *Human* represent the role's respective selectional restrictions — the types



```
(FRAME Inside-Of
  State
  :Roles (Obj (Physical-Obj 0.05)) (Loc (Container-Obj 0.3)) (Planner (Human 0.05))
  :Phrase       (S-is-inside-of-DO     1.0 (Obj Subject) (Loc Direct-Obj))
  :Result-Of    (Transfer-Inside       1.0 (Obj Obj) (Loc Loc) (Planner Actor))
  :Refinements  (Inside-Of-Dishwasher 1.0 (Obj Obj) (Loc Loc) (Planner Planner))
                (Inside-Of-Opaque      1.0 (Obj Obj) (Loc Loc) (Planner Planner))
                (Inside-Of-Carwash     1.0 (Obj Obj) (Loc Loc) (Planner Planner)))
```

Figure 1 Simplified definition of the frame representing the state *Inside-Of.*

of objects that are semantically allowed to bind to them. The rest of Figure 1 defines *Inside-Of*'s relations to other frames. The knowledge represented here is that it is (a) directly accessed by the phrase *S-is-inside-of-DO* (as in *"The fork is inside of the dishwasher"*), (b) a *result-of* the action *Transfer-Inside*, and (c) has several possible concept *refinement* frames: *Inside-Of-Dishwasher*, *Inside-Of-Opaque* and *Inside-Of-Carwash.*

Refinements (short for concept *refinement*s, an inverse of the *is-a* relation) of frames are useful because they allow specific inferences to be made when role-bindings are known (Lytinen [22]). For example, if the network has inferred that a cooking utensil is inside of a dishwasher (*Inside-Of-Dishwasher*), a likely inference is that it is about to cleaned. If the network has inferred that something is inside of an opaque object (*Inside-Of-Opaque*), the network can infer that it is blocked from sight. When multiple frames are defined as alternatives for a given relation to a frame, as in the multiple *refinement*s of *Inside-Of*, they are defined as *mutually exclusive* relations which compete for selection as the relation's instantiation at any given time. For example, although there are multiple possible *plans-for* the goal of *Satisfy-Hunger* (e.g., *Restaurant*, *Eat-At-Home*, etc.), generally only one will be used as the plan for a *given* instance of somebody wanting to satisfy his hunger in a particular story.

The relations and their role correspondences shown in Figure 1 also define the network's general knowledge rules, such as the following:

R3: (Subject X *S-is-inside-of-DO* DO Y) *phrase* (Obj X *Inside-Of* Loc Y)

(The phrase "X is inside of Y" means that object X is inside of object Y).

R4: (Actor X *Transfer-Inside* Obj Y Loc Z) *results-in* (Obj Y *Inside-Of* Loc Z Planner X)

(When actor X transfers an object Y into location Z, then Y is inside of Z).

Finally, the numbers in Figure 1 represent the connection weights (ranging from 0 to 1) from each of the related concepts to *Inside-Of*, and are chosen on the basis of how much evidence they provide. For example, if an object has just been transferred inside of something else (*Transfer-Inside*), then the network can definitely infer that the object is *Inside-Of* it. Therefore, the weight from *Transfer-Inside* to *Inside-Of* is maximal (1.0). If something that is a container (*Container-Obj*) has been mentioned in a story, then there is some, though not certain, evidence that something is inside of it, so a corresponding middling weight of 0.3 from *Container-Obj* to *Inside-Of*'s Location role is given. On the other hand, a very small weight (0.05) is given from *Physical-Obj* to *Inside-Of*'s Object role, since mere mention of any particular physical object does not very strongly imply *Inside-Of*. The actual weights chosen are clearly arbitrary. What is important is that they be in a *range* reflecting the amount of evidence the concepts provide for their related concepts in a certain knowledge base.

3.2 STRUCTURE OF ROBIN

The knowledge given to ROBIN is used to *construct* the network before any processing begins. As with other structured connectionist models, a single node in the network represents each frame or role. Relations between concepts are represented by weighted connections between the nodes. Activation on frame and role nodes is *evidential*, corresponding to the amount of evidence available from the current context for that concept. However, as described earlier, simply representing the amount of evidence available for a concept is not sufficient for complex inferencing tasks. Solving the variable binding problem requires a way to *identify* the concept that is dynamically bound to a role. Furthermore, the network's structure must allow such role-bindings to propagate across the network to dynamically instantiate inference paths and form an elaborated representation of the input.

3.3 VARIABLE BINDING WITH SIGNATURES

Representation of variables and role-bindings is performed in ROBIN by network structure that processes *signatures* — activation patterns that uniquely identify the concept bound to a role (Lange and Dyer [19]). Every concept in the network has a set of *signature units* that output its signature, a constant activation pattern different from all other signatures. A dynamic binding exists

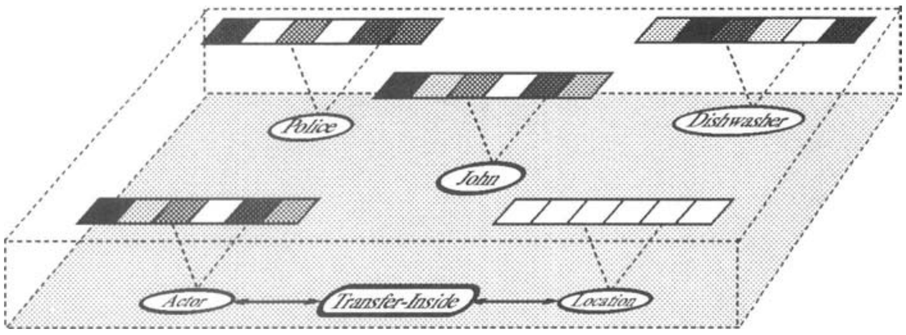


Figure 2 Examples of signature patterns (banks of units on top plane) for concepts (ovals on lower plan). Actor and Location roles and of the *Transfer-Inside* frame and their binding units are also shown.

when a role or variable's *binding units* have an activation pattern matching the activation pattern of the bound concept's signature.

An example of signatures is shown in Figure 2, which shows the concept nodes for the concepts *Police, John,* and *Dishwasher* (on the lower plane) and their associated signature units (banks of units on the top plane). Here signatures are shown as unique six-unit distributed patterns, with different levels of activation being represented by different levels of gray. The figure also shows some of the units for the frame *Transfer-Inside* and their activation values when its Actor is bound to *John.* The *virtual binding* of *Transfer-Inside*'s Actor role to *John* is represented by the fact that its binding units have the same activation pattern as *John*'s signature. The binding banks for the Location role have no activation because this role is currently unbound. The complete *Transfer-Inside* frame is represented in the network by the group of units that include the conceptual unit *Transfer-Inside,* a conceptual unit for each of its roles (the Object role not shown), and the binding units for each of its roles. The same binding units could, at another time, hold a different virtual binding, simply by having the activation pattern of another concept's signature.

In general, signatures can be uniquely-identifying activation patterns of any size. Ideally, signatures are distributed activation patterns (e.g., made up of semantic microfeatures) that are themselves reduced semantic representations of the concept for which they stand (as in Figure 2). Having the signatures represented as distributed activation patterns carrying semantic information may allow their future use as inputs for local distributed learning mechanisms after inferencing (discussed later). For simplicity, however, ROBIN's sim-

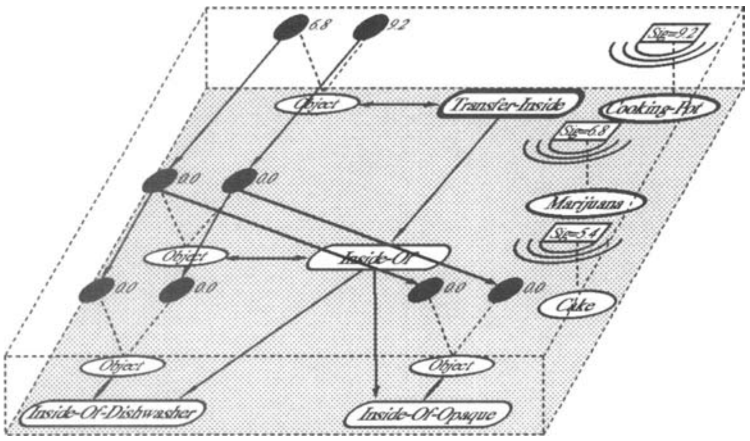


Figure 3 Simplified ROBIN network segment showing parallel paths of evidential activation (bottom plane) and signature activation (top plane). Initial activation is shown for *"John put the pot inside the dishwasher"*. Signature nodes (outlined rectangles) and binding nodes (solid black circles) are in the top plane. Thickness of conceptual node boundaries (ovals) in the bottom plane represents their levels of evidential activation. Node names do not affect the spread of activation in any way. Connections encode rule R4 and others.

ulations currently represent signatures as unique, arbitrarily-generated scalar values (e.g., 6.8 for *Marijuana* and 9.2 for *Cooking-Pot*).

3.4 PROPAGATION OF SIGNATURES FOR INFERENCING

The most important feature of signatures is that they can be propagated without change across long paths of binding units to dynamically instantiate inference chains. Figure 3 shows how the network's structure accomplishes this and automatically propagates signatures to fire rules (such as R4). Evidential activation for disambiguation is spread through the paths between conceptual units on the bottom plane, e.g., *Transfer-Inside* and its Object role. Signature activation for dynamic role-bindings and inferencing is spread across the parallel paths of corresponding binding units (solid black circles) on the top plane. As shown here, there are actually multiple binding units per role (e.g., the interchangeable left and right binding units of *Transfer-Inside*'s Object role). This allows simultaneous propagation of ambiguous bindings, such as the multiple meanings of the word *"pot"*. In general, this requires that each role have as many binding units as there are possible meanings of the network's most ambiguous word.

Initially there is no activation on any of the conceptual or binding units in the network. When input for a phrase such as *"John put the pot inside the dishwasher"* (***P1***) is presented, the lexical concept nodes for each of the words in the phrase are clamped to a high level of evidential activation. This directly provides activation for the concepts *John*, *Transfer-Inside*, *Cooking-Pot*, *Marijuana*, and *Dishwasher*. To represent the role-bindings given by phrase ***P1***, the binding units of each of *Transfer-Inside*'s roles are clamped to the signatures of the concepts bound to them (Actor and Location roles not shown). For example, the binding units of *Transfer-Inside*'s Object are clamped to the signature activations (6.8 and 9.2) of *Marijuana* and *Cooking-Pot*, representing the candidate bindings from the word *"pot"* (Figure 3)[2]. An alternative input, such as *"George put the cake inside the oven"*, would be represented by clamping the signatures of its bindings (i.e., *George*, *Cake*, and *Oven*) instead. A completely different set of inferences would then ensue.

The activation of the network's conceptual units is equal to the weighted sum of their inputs plus their previous activation times a decay rate, similar to the activation function of previous structured networks. However, the activation of the binding units is equal to the maximum of their unit-weighted inputs so that signatures can be propagated without alteration. Binding units calculate their activation as the maximum of their inputs because this preserves their signature input value even when the signature can be inferred from more than one direction. The actual relative signature activation values do not matter, because gated connections (not shown) ensure that two different signatures do not reach the same binding node.

As activation starts to spread after the initial clamped activation values in Figure 3, *Inside-Of* receives evidential activation from *Transfer-Inside*, representing the evidence that something is now inside of something else. Concurrently, the signature activations on the binding units of *Transfer-Inside*'s Object propagate to the corresponding binding units of *Inside-Of*'s Object (Figure 4), because each of the binding units calculates its activation as the maximum of its inputs. For example, *Inside-Of*'s left Object binding unit has only one input connection, that from the corresponding left Object binding unit of *Transfer-Inside*. This unit-weighted connection allows the network to make the inference that the Object of *Inside-Of* is the same as the Object of *Transfer-Inside* (from rule R4).

[2]ROBIN does not currently address the problem of deciding upon the original syntactic bindings, that is that pot is bound to the Object role of phrase ***P1***. Rather, their networks are initially given these bindings and then use them for high-level inferencing.

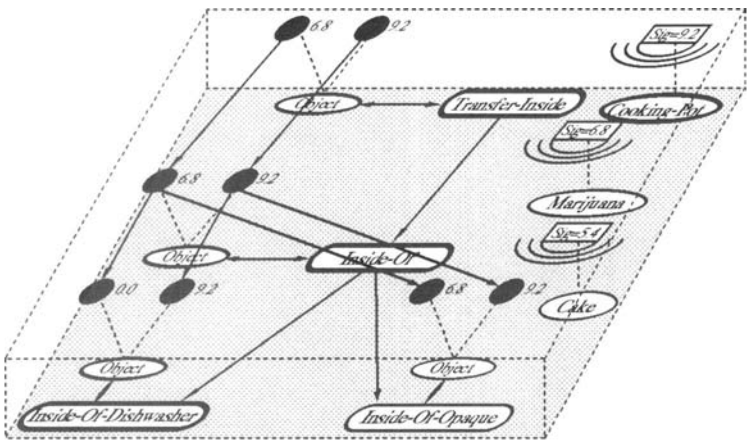


Figure 4 Simplified ROBIN network segment showing activation midway through processing ***Hiding Pot***. At this time, *Cooking-Pot* and *Inside-Of-Dishwasher* have higher evidential activations than *Marijuana* and *Inside-Of-Opaque*, as is illustrated by their thicker ovals.

Here, the left Object binding unit of *Transfer-Inside* has an activation of 6.8, so *Inside-Of*'s left Object binding unit also becomes 6.8 (*Marijuana*'s signature), because 6.8 is its maximum (and in this case only) input. The binding of *Cooking-Pot* (9.2) to *Inside-Of*'s right Object binding unit propagates at the same time, as do the bindings of *Inside-Of*'s Planner role to the signature of *John* and its Location to the signature of *Dishwasher* (not shown).

By propagating signature activations from *Transfer-Inside*'s binding nodes to *Inside-Of*'s binding nodes, the network has made its first inference. The network therefore not only holds the information that something is inside of something else (as shown by its evidential activation) — it also represents exactly which thing is inside the other through the signatures on its binding units.

ROBIN continues making inferences from the activations of this new knowledge in turn. Evidential and signature activation spreads, in parallel, from *Inside-Of* to its *refinements* *Inside-Of-Dishwasher* and *Inside-Of-Opaque* and their corresponding binding units (see Figure 4), on through the rest of the network[3]. Figure 5 shows an overview of the signature bindings in a portion of

[3]The reader may note that the signature for *Marijuana* (6.8) did not reach the left binding unit of *Inside-Of-Dishwasher* in Figure 4. This is due to additional structure of gated links that

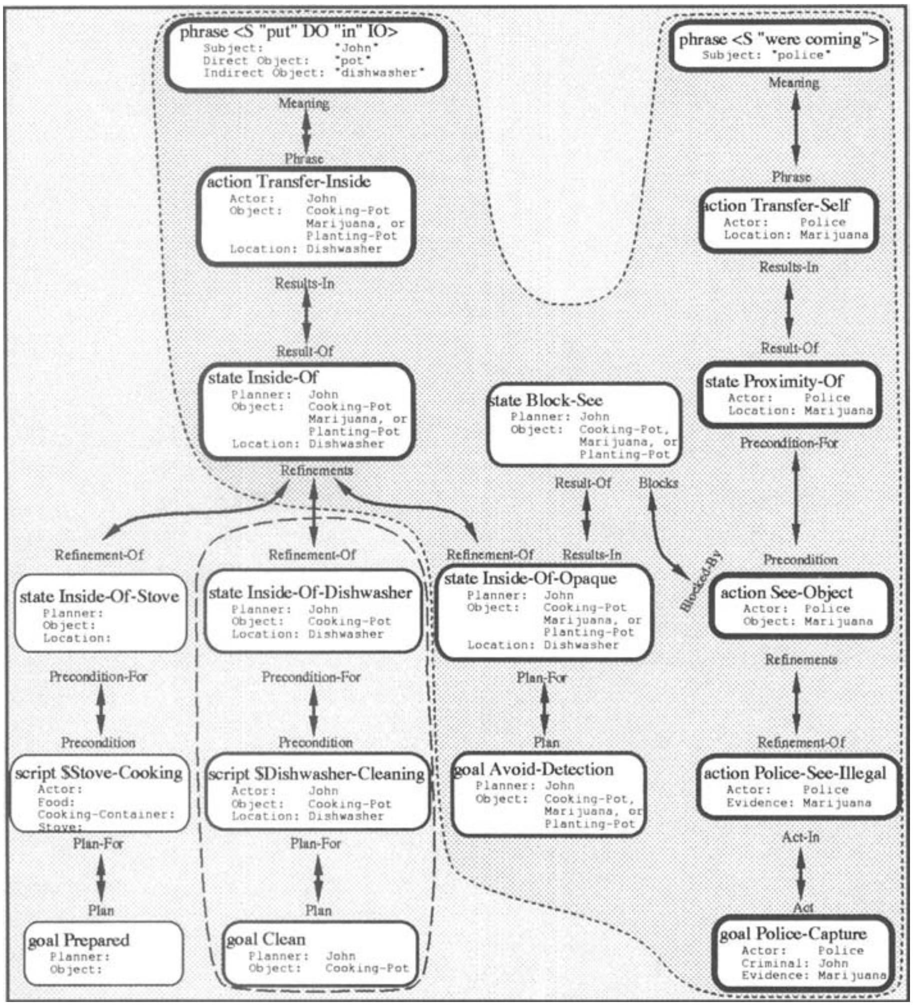


Figure 5 Overview of a small portion of a ROBIN network showing inferences made after clamping of inputs for the phrases of ***Hiding Pot***. Thickness of frame boundaries shows the amount of evidential activation on the frames conceptual nodes. Role fillers shown are the ones dynamically instantiated by propagation of signature activation over the roles' binding nodes (as in Figure 4). Darkly shaded area indicates the most highly-activated path of frames representing the network's interpretation of the input. Dashed area shows the discarded dishwasher-cleaning interpretation. Frames outside of both areas show a small portion of the rest of the network that received no evidential or signature activation.

the network after presentation of the input for the rest of ***Hiding Pot*** (*"because the police were coming"*) is presented and the network eventually settles. The network has made the inferences necessary to build the correct interpretation of the story in this domain, with most of the inferences being shown in the figure.

3.5 DISAMBIGUATION AND REINTERPRETATION

ROBIN's propagation of signature activations dynamically instantiates candidate inference paths in parallel in much the same way as marker-passing systems and the structured connectionist binding mechanisms of Shastri and Ajjanagadde [33] and Sun [35]. However, natural language understanding requires more than basic variable binding and rule-firing capabilities. It also requires the ability to resolve ambiguities and select between the large number of candidate inference paths instantiated by rule-firing. This is handled in ROBIN by the evidential activation that spreads in parallel with signature bindings.

If this were a marker-passing system constructing an internal representation of ***Hiding Pot***, it would need an external symbolic path evaluator to select between the cleaning path and the longer hiding path connecting *John*'s *Transfer-Inside* to the *Police*'s *Transfer-Self*. At the end of processing, the path evaluator would also have to recognize that *Marijuana* should be selected over the *Cooking-Pot* and *Planting-Pot* bindings throughout the network.

Such disambiguation is performed entirely within ROBIN's network without resorting to a separate path-evaluation module. Instead, the evidential portion of the network (e.g., the bottom plane of Figure 3) decides between the competing inference paths that have been instantiated by signature activation. The connections of the network and ROBIN's global inhibition mechanism (Lange and Dyer [19]) assure that the activations of the conceptual frame nodes are always approximately proportional to the amount of evidence available for them in the current context from their bindings and related frames. ROBIN's interpretation of its input is the most highly-activated path of frame units and their bindings when the network settles[4].

encode knowledge about what kind of concepts can be bound to the roles of particular frames — such that only concepts that are *refinements* of Cooking-Utensils are prototypically cleaned as the Object in *Inside-Of-Dishwasher*. These selectional restrictions and their importance are described in Lange and Dyer [19].

[4] As in all connectionist models, the network's "decision" or "selection" is actually simply the interpretation that the human modeller gives to the levels of activation present in it.

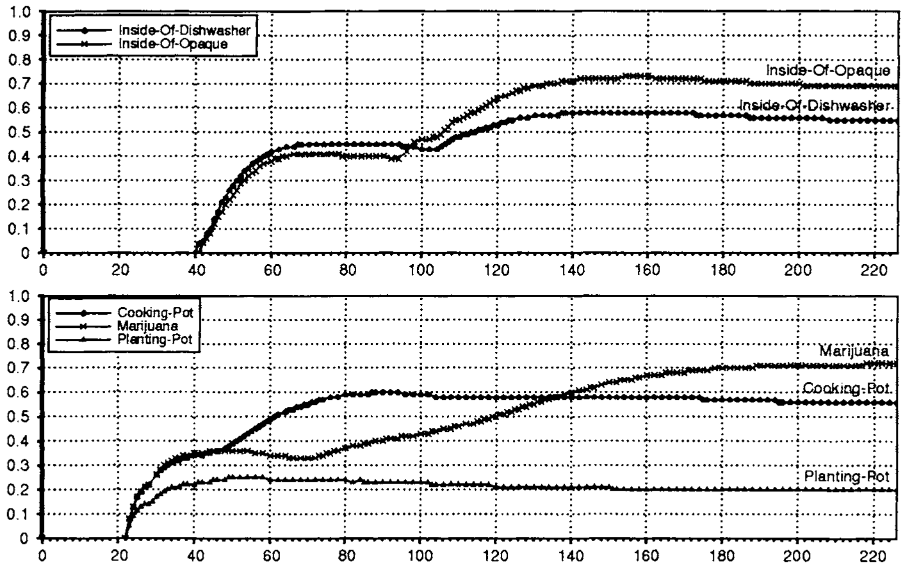


Figure 6 Evidential activations for meanings of *"pot"* and of competing *refinements* of *Inside-Of* after presentation of *"John put the pot inside the dishwasher"* at cycles 1 to 31 and *"the police were coming"* at cycles 51 to 61.

Often there are multiple possible competing interpretations for a given frame. This occurs when there are multiple meanings for a word or phrase, multiple plans to achieve a goal, or multiple *refinements* for a frame (e.g., the *Inside-Of-Dishwasher* and *Inside-Of-Opaque refinements* of *Inside-Of*). In these cases, the most highly activated interpretation that has been instantiated with compatible signature role-bindings is chosen as part of the inference path. The alternatives having lower activation are ignored, unless future context causes them to become more highly activated (leading to reinterpretation). Similarly, when there are multiple possible bindings for a role, the binding chosen at any given time is the one whose concept has the highest level of evidential activation.

Figure 6 illustrates how evidential activation works through constraint satisfaction to disambiguate meanings and interpretations. The evidential activations of the competing meanings of *"pot"* and *refinements* of *Inside-Of* change during the processing of ***Hiding Pot***. Initially there is more evidence for the interpretation that John was trying to clean a cooking pot. This is shown by the fact that after *Inside-Of-Dishwasher* becomes activated at about cycle 60, *Cooking-Pot* becomes more highly activated than *Marijuana* or *Planting-Pot*. Input for the second phrase of ***Hiding Pot*** (*"because the police were coming"*) is presented at cycles 51 through 61. The evidential activation levels shown by the thickness of conceptual node boundaries in Figure 4 correspond to the activations at cycle 90. The inferences about the police propagate through *Transfer-Self*, *Proximity-Of*, *See-Object*, and *Block-See*, until they reach *Inside-Of-Opaque* (see Figure 5). This occurs at about cycle 95. By about cycle 160, reinforcement from the *Block/See/Police-Capture* path causes *Inside-Of-Opaque* to become more activated than *Inside-Of-Dishwasher*, and *Marijuana* to become more highly activated than *Cooking-Pot*. Thus, ROBIN's interpretation of ***Hiding Pot*** is that John was trying to avoid detection of his *Marijuana* from the police by hiding it inside of an opaque dishwasher. The final inference path interpretation is shown in the darkly shaded area of Figure 5.

ROBIN's ability to use the constraint satisfaction of evidential activation in combination with its parallel dynamic inferencing with signatures makes it a promising approach to natural language understanding. Its automatic disambiguation abilities through evidential activation is a primary advantage over most symbolic marker-passing systems, which can also generate alternative inference paths in parallel, but which must use a serial path evaluator separate from the marker-spreading process to select the best interpretation, a significant problem as the size of the networks increase and the number of generated inference paths to be evaluated increases dramatically. More details of ROBIN's current abilities and structure are described in Lange and Dyer [19] and Lange [20]. Future research should expand these abilities further.

4 EPISODIC RETRIEVAL IN REMIND

Most research in analogical and case-based memory retrieval has explored retrieval in isolation from the comprehension process. We believe, however, that memory retrieval and comprehension are intricately related, and that much more can be learned by developing a model that integrates the two within a single mechanism. Building upon our work in ROBIN, we are developing REMIND, an inferencing-based model of memory retrieval that allows us to explore the effects of inferencing and disambiguation on the retrieval process. REMIND receives syntactic representations of short input texts as memory cues. Using general knowledge stored in its long-term memory, REMIND constructs elaborated interpretations of the cues, and then retrieves the episodes that are most similar to the surface and inferred features of their interpretations. This section provides an overview of how REMIND works. A more detailed description can be found in Lange and Wharton [21].

REMIND uses the same simple spreading-activation mechanism as ROBIN to encode world knowledge and perform inferencing and interpretation (see section 3). REMIND's networks also contain representations of prior episodes, such as *"Fred put his car in the car wash before his date with Wilma"* (***Car Wash***) and *"Billy put his Playboy under the bed so his mother wouldn't see it and spank him"* (***Dirty Magazine***). The representations of these episodes are the actual plan/goal analyses (or interpretations) that the network inferred for them earlier. These prior episodes are indexed into the comprehension network through connections with the knowledge structure nodes of their representation.

To perform retrieval, REMIND is given a short text passage to use as a deliberate memory cue, such as *"John put the pot inside the dishwasher because the police were coming"* (***Hiding Pot***). Evidential and signature activation spread through the ROBIN portion of the network to disambiguate and infer an interpretation of the cue (as described in section 3). Because the units representing long-term memory episodes are connected within the network, an important side-effect of this understanding process is that episodes having concepts related to the elaborated cue also become highly activated. This includes episodes related due to superficial semantic overlap with the cue (e.g., other episodes involving police, drugs, or kitchen appliances) and episodes related abstractly because they share similar inferred plans and goals of their actors (e.g., episodes that share the inferences that a person was trying to *Avoid-Detection* of something to avoid *Punishment*, such as ***Dirty Magazine***). After the network settles, the episode that received the most activation from the cue's interpretation and surrounding context becomes the most highly activated, and is therefore retrieved as the best match for the cue.

4.1 Representation of Long-Term Episodes

Whereas the general world knowledge and inference rules used to initially build REMIND's networks are hand-coded, REMIND is not given any information about the particular episodes it is going to understand and store in long-term memory. The representations used for these target episodes are created entirely by REMIND's spreading-activation understanding process. Input for each episode's text is presented to the network, which then infers an interpretation of it by the spread of signature and evidential activation. Next, units and connections are added (by hand) to store the episode's entire resulting interpretation in REMIND's long-term memory. Accordingly, each episode's representation includes all aspects of its interpretation, from its disambiguated surface features (such as the actors and objects in the story) to the plans and goals that REMIND inferred that the actors were using. The units added to encode long-term episodes' representations are added to the same network used for the inferencing and understanding process, causing both processes to interact with and affect each other.

As a complete example, consider how ***Dirty Magazine*** (*"Billy put the Playboy under his bed so his mother wouldn't see it and spank him"*) is processed and stored in the network as a memory episode. First, signature and evidential activations representing its phrasally-analyzed input are clamped to start the understanding process. As described earlier for ***Hiding Pot***, the input is presented to the network by clamping the evidential activations of the input's phrase and word nodes to 1 and clamping the binding units of the phrases' roles to the signatures of their bindings' word meanings. Activation then spreads through the network to infer and disambiguate an interpretation of the input.

As in ***Hiding Pot***, the network infers that somebody is hiding something (*Avoid-Detection*) and that it is blocked from sight (*Block-See*). Here, however, the inferred signatures show that it is *Billy* hiding a *Playboy-Magazine* rather than *John* hiding *Marijuana*. Several other knowledge structures involved in ***Hiding Pot*** (e.g., *Proximity-Of*, *Possess-Obj*, *Punishment*) are also activated by ***Dirty Magazine***. These similarities make ***Dirty Magazine*** a likely candidate for reminding when the network is presented with ***Hiding Pot*** as a cue. However, there are a number of differences, e.g. frames of the *Guardian-Discipline* structure are part of ***Dirty Magazine***'s interpretation, but the *Police-Capture* frames are not.

Figure 7 shows an overview of the network after ***Dirty Magazine*** and several other episodes (from Figure 8) have been understood and memorized. The frames activated as part of ***Dirty Magazine***'s interpretation are shown by nodes that have a circled "1" above them in the figure. Other circled numbers represent elements of other stored episodes' interpretations. It is important to note that each episode's representation also includes all of the simple bridging inferences that were necessary to make the plan/goal analysis. Here the bridging inferences for ***Dirty Magazine*** include that the Playboy was *Under.1* the bed, that Billy possessed the Playboy (*Possess-Obj.1*), that the salient *refinement* of this possession was that it was possession of a naughty object (*Possess-Naughty-Obj.1*), and so on.

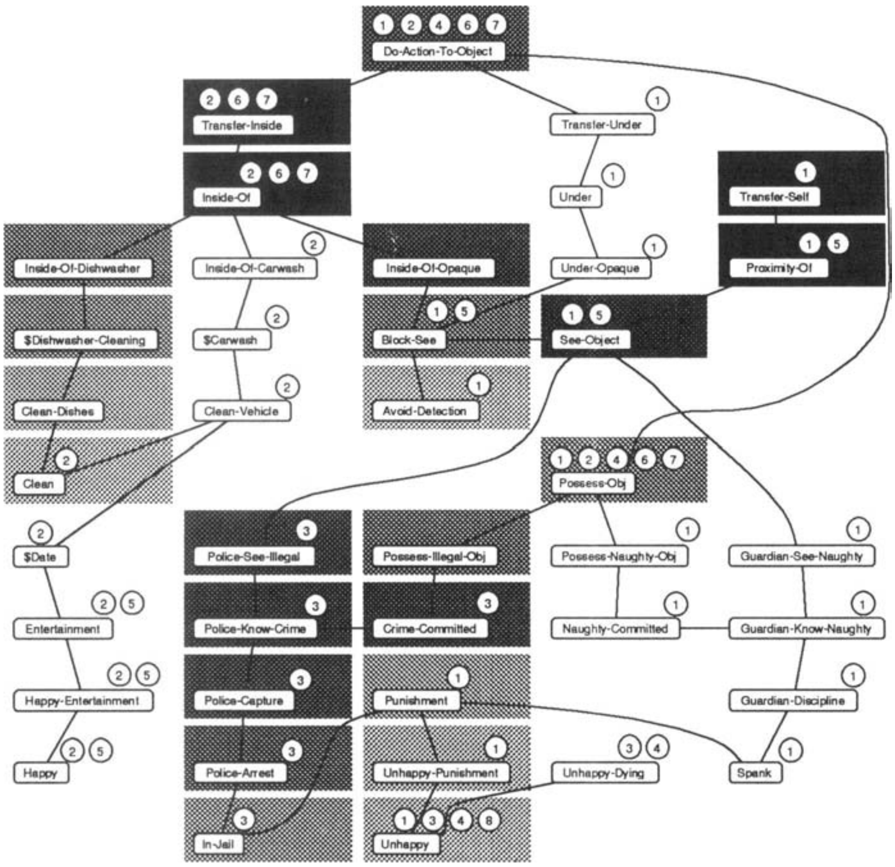


Figure 7 Overview of part of a REMIND network storing the episodes of Figure 8. Circles above frames indicate long-term instances connected to them. Numbers within circles indicate which episode the instance is part of. Overview is shown after activation has settled in processing of ***Hiding Pot.*** Gray boxes around nodes represent the final level of evidential activation on the frame concept nodes (darker = higher activation, no box = no activation).

#	Episode Text	Phrasally Parsed Input Given the Network
1	*Billy put the Playboy under his bed so his mother wouldn't see it and spank him.* **(Dirty Magazine)**	(<S-put-DO-under-IO> (S Billy) (DO Playboy) (IO bed)) (<S-see-DO> (S Mother) (DO Playboy)) (<S-spank DO> (DO Billy))
2	*Fred put his car inside the car wash before his date with Wilma.* **(Car Wash)**	(<S-put-DO-inside-IO> (S Fred) (DO car) (IO carwash)) (<S-date DO> (S Fred) (DO Wilma)
3	*Jane shot Mark with a Colt-45. He died.*	(<S-shot-DO-with-IO> (S Jane) (DO Mark) IO Colt-45)) (<S-died>) (S Mark))
4	*Betty wanted to smoke a cigarette, so she put it on top of the stove and lit it.*	(<S smoke DO> (S Betty) (DO cigarette)) (<S put DO on top of IO> (S Betty) (DO cigarette) (IO stove)) (<S lit DO> (DO cigarette))
5	*The pleasure boat followed the whales to watch them.*	(<S followed DO> (S pleasure-boat) (DO whales)) (<S watch DO> (DO whales))
6	*Barney put the flower in the pot, and then watered it.* **(Flower Planting)**	(<S put DO inside IO> (S Barney) (DO flower) (IO pot)) (<S watered DO> (DO flower))
7	*Mike was hungry. He ate some fish.*	(<S was hungry> (S Mike)) (<S ate DO> (S Mike) (DO fish))
8	*Suzie loved George, but he died. Then Bill proposed to her. She became sad.* **(Sad chapter)**	(<S loved DO> (S Suzie(DO George)) (<S died> (S George)) (<S proposed to IO> (S Bill) (DO Suzie)) (<S became sad> (S Suzie))

Figure 8 Episodes understood and stored in the network of Figure 7.

Once the full interpretation for an episode has been determined, units and connections representing it are hand-coded into the network's long-term memory. For ***Dirty Magazine***, the units added include (a) nodes representing each instantiated frame of its interpretation in Figure 7 (e.g., *Billy.1*, *Playboy-Magazine.1*, *Avoid-Detection.1*, and *Possess-Obj.1*), (b) units to represent their roles, and (c) a unit to stand as a placeholder for the entire episode (e.g., *Episode.1*). These units are then connected to their corresponding local elements in the normal evidential semantic network. They are also interconnected to encode their role-bindings and which episode they are part of.

Figure 9 shows an example of the units and connections that are added to the network to represent episodes. The figure shows a simplified part of the network's evidential layer after several episodes have been understood and added to long-term memory. The gray units in the figure are the normal semantic conceptual units originally in the network, including the conceptual units for frames *Possess-Obj* and *Possess-Naughty-Obj* and a number of other frames (shown here as connected in a *refinement* is-a hierarchy). At this stage, two episodes have been processed that include *Possess-Obj* or *Possess-Naughty-Obj* as part of their interpretation: ***Dirty Magazine*** (*Episode.1*), and *"Betty wanted to smoke a cigarette, so she put it on top of the stove and lit it"* (***Cigarette Lighting***; *Episode.4*). ***Cigarette Lighting***'s interpretation

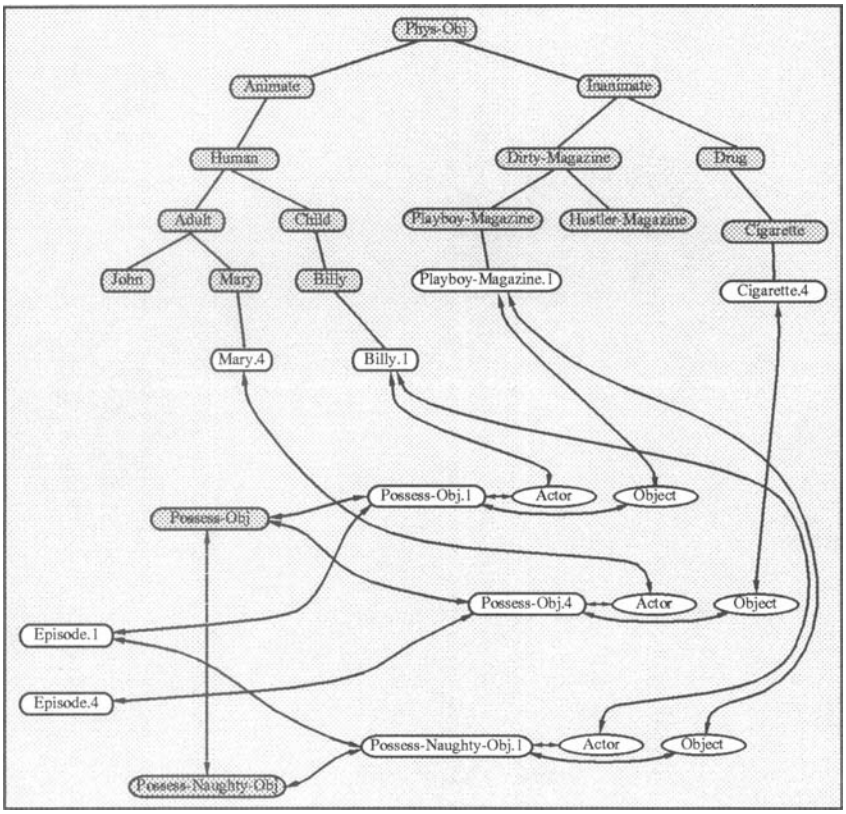


Figure 9 Encoding of *Possess-Obj* and *Possess-Naughty-Obj* instances for *Episode.1* (***Dirty Magazine***) and *Episode.4* (***Cigarette Lighting***). Gray units are pre-existing conceptual nodes.

includes an instance of *Possess-Obj* because the network inferred that Betty must possess the cigarette to light it.

The white units in Figure 9 show some of the units added to the network to encode ***Dirty Magazine*** and ***Cigarette Lighting***. For each episode, there is a single *episode unit* serving to represent and group all of its elements together, such as *Episode.1* and *Episode.4* in Figure 9. In addition, there is an *episode instance unit* representing each element of the episode's interpretation. For ***Dirty Magazine***, there is an episode instance unit for *Billy.1*, *Playboy-Magazine.1*, *Possess-Obj.1* and *Possess-Naughty-Obj.1*, along with units (not shown) representing all of the other elements of its representation. These episode instance units are connected both to the general semantic concept they instantiate (e.g., *Billy.1* is connected to *Billy*) and to the episode unit of which they are part (e.g., *Episode.1* for ***Dirty Magazine***'s elements). Furthermore, each episode instance is connected to units representing its roles (e.g., the Actor and Object unit for *Possess-Obj.1*), which are in turn connected to the concepts that were bound to them (e.g., *Possess-Obj.1*'s Actor is connected to *Billy.1*, and its Object is connected to *Playboy-Magazine.1*). The rest of the interpretation of each episode (e.g., in Figure 7) is encoded similarly with units and connections that represent all of its other instantiated frames and elements.

As can be seen, REMIND's method of encoding its episodes is different from that of many memory retrieval and case-based reasoning models. Episodes in REMIND are not indexed under any one knowledge structure or important groups of knowledge structures. They are instead indexed under every concept that was an aspect in understanding them in the first place. These concepts include both the abstract inferences that make up the plan/goal analysis of the episode *and* the simple disambiguated surface semantic features of the text (such as its direct word and phrase meanings). This fully dispersed form of indexing has important implications for the kinds of remindings that the model produces. Notice that more specific frames tend to have fewer episode instances than less specific frames (see Figure 7). This is to be expected, since specific knowledge structures pertaining to certain situations (such as a police search or a parent disciplining a child) represent events that are less frequently encountered than general knowledge structures about simple actions and states (such as being inside of something, or possessing an object). As an example, five of the episodes in Figure 7 (1, 2, 4, 6, and 7) inferred a *Possess-Obj* as part of their interpretation, but only one episode (1) involved a *Possess-Naughty-Obj* or *Avoid-Detection*. An important consequence of specific frames providing activation evidence for a smaller number of instances is that specific, contentful knowledge structures tend to be stronger reminding indices than general ones.

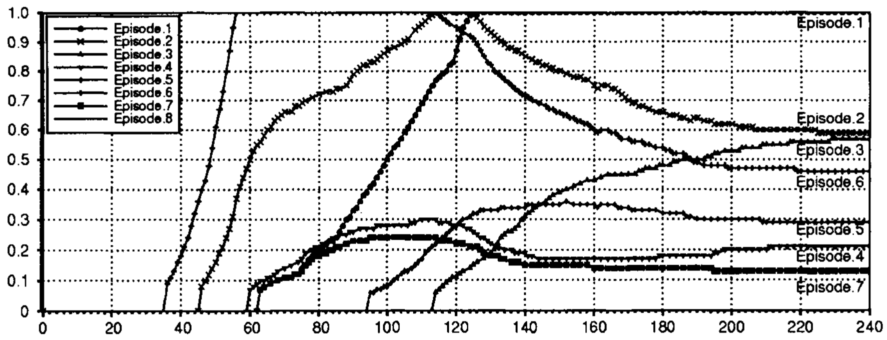


Figure 10 Evidential activations of episode units for eight episodes of Figure 8 after presentation of ***Hiding Pot***. *Episode.6*'s activation reaches asymptote (1.0) at around cycle 56, but declines at around cycle 112, when *Episode.2* reaches 1.0, until *Episode.2* declines again when *Episode.1* reaches 1.0 at around cycle 124.

4.2 The Process of Episodic Reminding

Retrieval in REMIND begins with presentation of an input cue to the network to be understood. Because episode instance units are connected directly to their corresponding concept units in the same network, they become active when the concepts they are instantiations of become activated by the understanding process. The more similarities an episode shares with the inferred interpretation of a cue, the more of the episode's instance units will become active. Episodes having a number of elements in common with the cue's interpretation therefore tend to become highly active. After the network settles, the episode with the most highly-activated episode unit is retrieved.

Now consider what happens when input for ***Hiding Pot*** is presented as a cue to the network. Evidential and signature activation spread through the network, dynamically instantiating the competing inference paths as described earlier. At the same time, similar episodes that are connected to those inferences through their episode units also become activated. Figure 10 shows the activation levels of the eight episodes as activation spreads through the network. As can be seen, *Episode.6* (*"Barney put the flower in the pot, and then watered it"*) initially becomes highly active because it shares a number of surface features with ***Hiding Pot***. For example, both involve a *Transfer-Inside*, both have humans, and *Planting-Pot* receives activation from the word *"pot"*. Similarly, *Episode.2* (***Car Wash***) initially becomes active because of shared surface features with ***Hiding Pot***. *Episode.2*'s activation continues to climb when the *Clean* frame is inferred, since a *Clean* is part of ***Car Wash***'s interpretation. However, as REMIND continues to process ***Hiding Pot***, the hiding and punishment frames are inferred and become active. Eventually, *Episode.1*'s (***Dirty Magazine***) activation climbs and wins because it shares the most surface *and* abstract features of any episode with ***Hiding Pot***'s interpretation (see Figure 7). ***Dirty Magazine*** is therefore retrieved as the episode most similar to ***Hiding Pot***.

An explanation for why ***Dirty Magazine*** becomes the most-highly activated of the eight episodes can be seen in Figure 7. The gray boxes around nodes in Figure 7 indicate the final levels of evidential activation of the frames inferred for ***Hiding Pot***. Of the eight episodes stored in the network, ***Dirty Magazine*** has the most instances of its interpretation shared with ***Hiding Pot***'s final active interpretation (e.g., instantiations *Avoid-Detection.1*, *Block-See.1*, *Punishment.1*, and *Possess-Obj.1*). It therefore eventually becomes the most activated of the episodes.

Besides serving as an example of retrieval in REMIND, this example illustrates a number of important points about the model. The first point to notice is that even when the network settles, the losing episodes retain significant activation — enough that they could potentially be recalled later. As in the ROBIN portion of the network, this is the result of controlling episodes' activations through REMIND's global inhibition mechanism. The mutual (or competitive) inhibition mechanism used to control the spread of activation in many structured connectionist models drives the activation of "losing" interpretations down to zero, making reinterpretation difficult or impossible. In contrast, ROBIN and REMIND use a global inhibition mechanism that inhibits all evidential units by an equal damping factor. This allows competing frames and episodes to retain a level of evidential activation relative to the amount of evidence available for them, facilitating reinterpretation if warranted by later context.

A second point of interest is that elements and episodes that are superficially similar to the cue tend to become activated *before* elements and episodes that are only abstractly related to the cue (through inferences). This is a direct result of the spreading-activation process, since activation and signature inferences reach closely-related concepts before they reach more distant concepts. An example of this was seen in Figure 10, where the superficially-related *Episode.6* became activated before the more abstractly-related ***Dirty Magazine***. As seen, however, the early activation of superficially-similar episodes does not stop abstractly-similar episodes from winning if the abstractly-similar episodes ultimately share more features and activation with the cue. Because all episodes retain their relative supported levels of activation, abstractly-similar episodes such as ***Dirty Magazine*** can climb as inferences reach them and end up with the highest level of activation when the network settles.

Another important thing to note is that retrieval in REMIND is not all-or-nothing. As in human recall, REMIND often gets *partial* recall in which only subparts of the retrieved episode are activated. Parts of the retrieved episode distant from the current context of inferences may not become activated initially. This is true, for example, for parts of ***Dirty Magazine*** that differ significantly from ***Hiding Pot***'s interpretation (such as the *Guardian-Discipline* and *Spank* structures, which are relatively distant from anything in ***Hiding Pot***). However, the directly similar inferences between episodes and their primary actors and objects in episodes, such as *Billy.1* and *Playboy-Magazine.1* in ***Dirty Magazine***, do tend to become active because they play a part in so many of its roles. A final aspect to note about REMIND is how its language understanding and retrieval processes come full circle. The episode retrieved depends crucially on the interpretation of the cue from the spreading-activation network's inferences. Once an episode is retrieved, it in turn primes the activation of the evidential spreading-activation network, perhaps leading to a different disambiguation and therefore interpretation of the next cue.

Theoretically, REMIND lies somewhere between case-based reasoning models and general analogical retrieval models such as ARCS and MAC/FAC. Like ARCS and MAC/FAC, REMIND is meant to be a psychologically-plausible model of general human reminding, and therefore takes into account the prevalence of superficial feature similarities in remindings. However, we believe that many of the types of high-level planning and thematic knowledge structures used as indices in case-based reasoning systems also have an important effect on reminding. REMIND is thus partially an attempt to bridge the gap between case-based and analogical retrieval models. As it turns out, this gap is naturally bridged when the same spreading-activation mechanism is used to both understand cues and retrieve episodes from memory. Using the same mechanism for both processes causes retrieval to be affected by all levels that a text was understood with. This is the case in REMIND, in which the understanding mechanism is given the superficial features and actions of a text and attempts to explain them by inferring the plans and goals being used — causing memory episodes to be activated by both. This seems to give a more psychologically-plausible form of reminding than previous models, because the episodes it retrieves have varying degrees of superficial and abstract similarities to the cue (Lange and Wharton [21]; Wharton and Lange [41]), as seems to be the case in human reminding.

5 FUTURE WORK

Our initial research on developing structured connectionist models for natural language understanding and episodic memory retrieval seems quite promising. The use of signatures in ROBIN allows it to perform some of the variable binding and parallel dynamic inferencing difficult for connectionist models, while its integration within the constraint satisfaction abilities of its evidential layer allows it to perform disambiguation and reinterpretation difficult for traditional symbolic models. In turn, REMIND has illustrated many of the potential computational and predictive benefits of integrating the language understanding and episodic memory retrieval processes.

However, ROBIN and REMIND's representation and rule-firing abilities are currently limited relative to those of traditional symbolic models, limiting the length and complexity of the texts the model can understand and remember. Here we discuss planned advances to ROBIN and REMIND to improve their capabilities and allow us to explore how well their initially promising results scale up to more complex and longer inputs and texts. These include: (1) the ability to handle multiple dynamic instances of each frame, (2) the ability to handle more complex rules, (3) signatures as distributed patterns of activation to allow learning, and (4) the ability to handle more complex rules having conjunctive terms. We plan to integrate these new inferencing abilities both into ROBIN and REMIND and to perform a number of experiments on how they allow the models to scale up.

5.1 MULTIPLE DYNAMIC INSTANCES

One of the main restrictions of the model as described is that there can be only one dynamic instance of each frame at any given time, since binding units can only hold one signature at once. Because of this, ROBIN cannot yet represent or interpret texts involving two different seeing or eating events, for instance. We plan on developing a solution in which each concept in the network is actually represented by a small number of separate sub-networks of conceptual and binding units that can each represent a single dynamic instance of the conceptual frame with signatures. This will increase the size of the network by a linear amount k equal to the average number of dynamic instance sub-networks per concept, but will allow processing of inputs that involve multiple dynamic instances of the same frame, as in a similar approach for phase-binding networks described by Shastri and Ajjanagadde [33]. An important issue to be resolved in this approach is how to connect and gate the

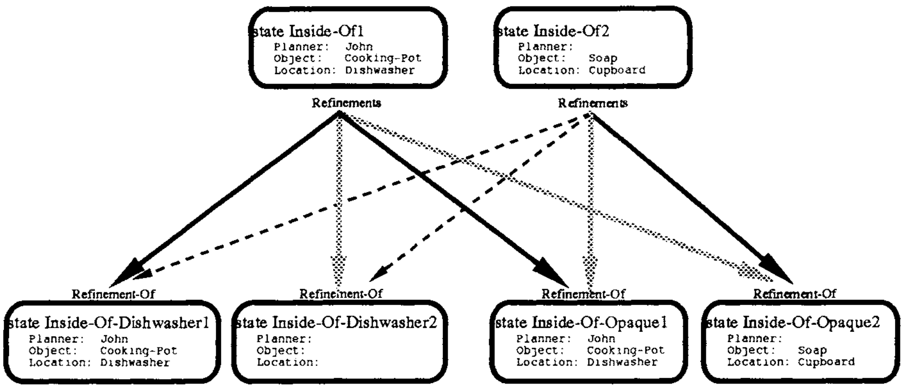


Figure 11 Overview of nework handling multiple dynamic instances of every frame. Black lines indicate paths over which signatures have passed. Dashed lines indicate paths ruled out because they violate the frame's selectional restrictions. Grey lines indicate paths ruled out because a dynamic instance already exists.

sub-networks so that the proper inferences are made, while assuring that the parallel evidential portion of the network is immune to crosstalk.

Figure 11 illustrates this solution and some of the issues involved. It shows an overview of a portion of the network having multiple dynamic instances per frame. Here each frame has two separate sub-networks, each potentially holding the signatures and evidential activation for one dynamic instance. *Inside-Of*, for example, is shown with two separate dynamic instance sub-networks: *Inside-Of1* and *Inside-Of2*. The overview shows a hypothetical state of the network after processing input for *"John put the pot inside the dishwasher, but the dishwasher soap was inside the cupboard"*. The sub-network for *Inside-Of1* holds the instance of the *Cooking-Pot* inside of the *Dishwasher*, while *Inside-Of2* holds the instance of *Soap* inside *Cupboard*. Each dynamic instance sub-network will have the same structure of conceptual and binding units as ROBIN does now to hold a single dynamic instance (cf. Figure 3).

Each dynamic instance sub-network of a frame uses the same basic network structure as is currently used for a single instance. However, there must be additional structure for the connections *between* related frames as defined by the knowledge base's general knowledge rules. This multiple instance gating structure will be responsible for controlling the spread of activation to particular dynamic instance sub-networks as inferences are made. The first thing this multiple instance gating structure must do is to assure that only *one* instance of a related frame receives signatures when an inference is made. Consider what should happen in Figure 11 if there is no activation in the network until *Inside-Of1* gets activated as a *Cooking-Pot Inside-Of* a *Dishwasher*. Signature and evidential activation should then spread to *Inside-Of-Dishwasher* and *Inside-Of-Opaque*, since they can both be inferred from it. However, only one dynamic instance of each should be inferred. The multiple instance gating structure between the frames must therefore assure that *Inside-Of1*'s activation only propagates to *Inside-Of-Dishwasher1* and *Inside-Of-Opaque1*, respectively, and not to either *Inside-Of-Dishwasher2* or *Inside-Of-Opaque2*. This is shown in Figure 11 by the grey (gate closed) connections from *Inside-Of1* to *Inside-Of-Dishwasher2* and *Inside-Of-Opaque2*.

The multiple instance gating structure must also assure that activation does not propagate to instance sub-networks that already hold instances. It should instead channel it to the first available instance sub-network. For example, when *Inside-Of2* (*Soap Inside-Of* a *Cupboard*) gets activated in Figure 11, its activation should not propagate to *Inside-Of-Opaque1* (grey arrow), which is already filled with a *Cooking-Pot* inside of a *Dishwasher*. Its activation and signatures should instead propagate to the first available sub-network, *Inside-Of-Opaque2*. Notice that this multiple instance gating structure must interact with the other structure in the network that enforces selectional restrictions by stopping activation from propagating when binding constraints are violated. For example, even though the *Inside-Of-Dishwasher2* sub-network is free to receive activation, it should not receive activation from *Inside-Of2*'s instance, since *Soap* inside of a *Cupboard* violates the constraints on *Inside-Of-Dishwasher* (that a *Cooking-Utensil* be inside of a *Dishwasher*).

The actual multiple instance gating structure will be similar to the structure currently in ROBIN for enforcing selectional restrictions (i.e. assuring that only legal inferences are made). This will be done with the same sort of simple units and connections to compare activations as in the rest of the model, with additional small winner-take-all networks to break ties when two different instances arrive at the same time.

5.2 MORE COMPLEX RULES

Using signatures of pre-existing concepts, ROBIN can create and infer novel network instances. However, ROBIN currently only propagates signatures

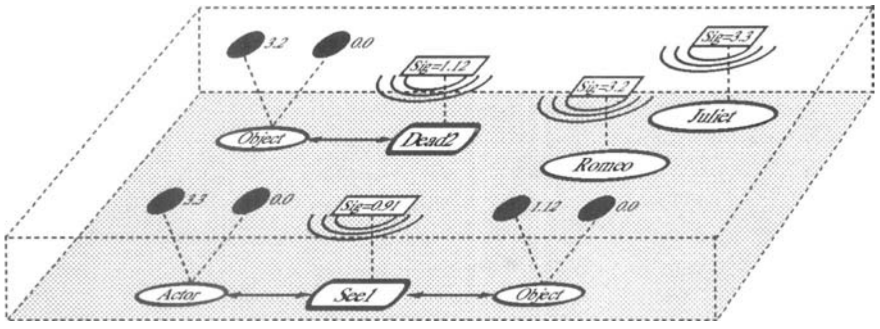


Figure 12 Example of using the signature of a dynamically-created instance.

of *pre-existing* concepts, such as of *Cooking-Pot*, *Marijuana*, or *John*. It does not propagate signatures of the *dynamically* created instances inferred by signatures (e.g., the dynamic instance of *Cooking-Pot* or *Marijuana* being *Inside-Of* a *Dishwasher* in Figure 4). Inferences using the dynamic instances themselves as bindings are necessary to encode most rules for *general* planning knowledge or complex interactions of goals, which generally require the ability to reason over any dynamic plan or goal instance the system might have. It is also crucial that the network be able to encode rules using combinations of terms (such as conjunctive terms). All of these types of rules are needed to understand many complex texts, such as those involving abstract planning failures or themes (cf. Schank [30]; Dyer [16]). These types of relatively complex rules are particularly difficult problems for connectionist models.

The first thing the network must be able to do is to hold and propagate signatures representing the dynamic instances created when sub-networks representing a frame are instantiated with signatures. For example, in the sentence *"Juliet saw that Romeo was dead"*, Juliet did not see a pre-existing person or thing. She saw a new state — that Romeo was dead. This new state instance itself is easily represented by the signature for *Romeo* being placed on one of the binding units of the Object role of the first available *Dead* frame instance (e.g., *Dead2*). The difficult part, however, is that the network must somehow be able to represent and propagate new instances such as this as signatures.

A solution to this problem is to use the frames' signatures themselves to represent the dynamic instance they hold. Figure 12 shows an example of how this can be done. *Dead2* has the activation representing the instance of Romeo being dead. The signature of *Dead2* shown here is 1.12. The fact that Juliet saw this is represented by binding her signature (3.3) to one of the binding units of *See1*'s Actor, and binding one of the binding units of its Object to the signature of *Dead2* (1.12). The network therefore dynamically represents the fact that *Juliet* sees some concept, which happens to be the instance held by the signature of *Dead2* (that *Romeo* is dead). The same signature (1.12) would be used during processing if at some other time *Dead2* held another dynamic instance, such as that *President-Hoover* was dead. Using the pre-existing signature of each frame instance sub-network will allow ROBIN to hold and propagate signatures representing dynamic instances. For example, in this case, the network would have a rule that said that if somebody *See*s an existing state, then they *Know* it. The network would then infer that Juliet *Know*s that Romeo is dead by propagating its signature (1.12) from *See1* to *Know1*.

It is also important to extend ROBIN's network structure to handle more complex rules that themselves create and propagate new instances that are functions of their instances, as opposed to just the signatures themselves. In predicate logic terms, this is the same as admitting rules with function terms. Consider, for instance, a simple rule that says "If somebody punches somebody else, then that person's nose will be broken," or:

R5: (Actor X *Punch* Object Y) *results-in* (*Broken* Object (*Nose* Y))

As with ROBIN's normal rules, R5 would spread activation to its consequent frame, *Broken*. But instead of causing *Broken*'s Object role to be bound to one of the concepts that *Punch*'s roles were bound to (i.e. *X* or *Y*), it will cause it to be bound to a *function* of one of those concepts — the *(Nose Y)*. So R5 should cause two things to happen. First, it should cause activation to spread to create a new instance of *Nose* whose Owner is *Y*. Second, it should make *Broken*'s Object role receive the signature of that new instance of *Nose*.

Figure 13 shows how the network will do this. It shows the evidential and binding unit sub-networks representing one instance each of *Punch* (*Punch1*), *Broken* (*Broken1*), and *Nose* (*Nose1*). The Object binding units of *Punch1* has connections directly to the Owner binding units of *Nose1*, since it is the Object of the *Punch*ing (*Y*) whose nose is involved. There is then a connection directly from the signature of *Nose1* to one of the Object binding units of *Broken1*, since the network can infer that *Nose1* will be broken. The signatures show the network after it has propagated activation for *"Juliet punched Romeo"*. The network has inferred that the concept whose signature is 4.21 is broken, where 4.21 is the signature of *Nose1*, representing *Romeo* (9.2)'s nose.

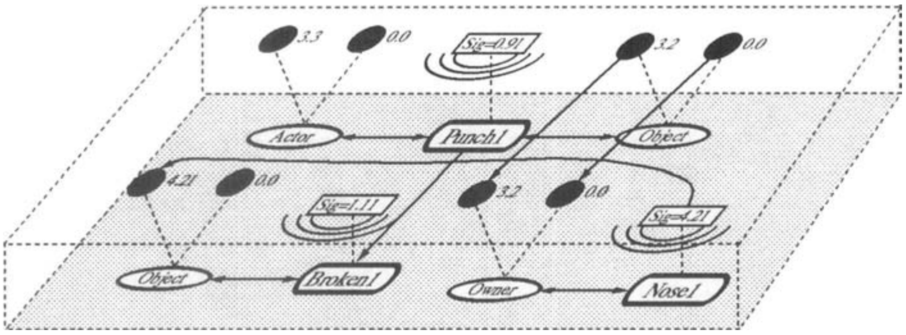


Figure 13 Network structure encoding a rule with a function term (R5).

A number of issues need to be resolved to determine the structure needed to allow rules with function terms. As with normal rules, signature and evidential activation should only propagate when the selectional restrictions (binding constraints) are not violated and when there is enough evidential activation to support the inference. Thus, in Figure 13, the connection from the signature unit of *Nose1* should have a gate that only allows the signature to propagate to the first Object binding unit of *Broken1* when the inference can be made. This structure will have to work consistently with the structure allowing multiple dynamic instances of every frame described in the previous section. Similarly, connections between concepts on the evidential layer will have to be gated so that they only propagate evidential activation when an inference can be made.

Similar changes need to be made to allow ROBIN to handle rules with conjunctive antecedents, where more than one thing must be true for the rule to fire. For example, a rule for jealousy might be that one person (*X*) is jealous of a second person (*Z*) if they love somebody (*Y*) *and* that person (*Y*) loves the second person (*Z*). To encode such rules, the structure of the network must only allow signature and evidential activation to propagate to instantiate a resulting instance when all antecedents are active and all of their bindings meet the rules' constraints (such as that the signature of the person that *X* loves is the same as the signature of the person *Y* who loves somebody else). The same kind of network structure encoding other kinds of binding constraints and comparing signatures to the expected signatures can be simply extended to handle these kinds of conjunctive rules.

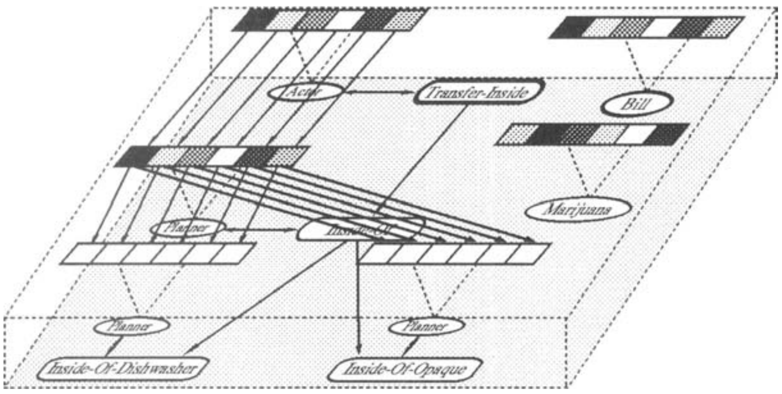


Figure 14 Distributed signatures, where each signature is a unique pattern distributed over a bank of units. Here each signature or binding bank is made up of six units, with increasing levels of activation represented by increasing darkness of shading (ranging from white = 0 to black = 1). Shown is the (desired) state of the network after *Bill*'s distributed signature has propagated from the binding bank of *Transfer-Inside*'s Actor to the binding bank of *Inside-Of*'s Planner, but before reaching *Inside-Of-Dishwasher* and *Inside-Of-Opaque*.

5.3 DISTRIBUTED SIGNATURES

Currently, each signature is a single arbitrary scalar value that uniquely identifies its concept. Large models could conceivably have thousands or hundreds of thousands of separate concepts that they could recognize (such as *Marijuana, Cooking-Pot, Catfish, Guppy, John, John-Wayne, John-Kennedy*, etc.). It is untenable to expect a single binding node to have enough precision to accurately distinguish between such a large number of signatures[5].

A better solution is that each signature be a *distributed* pattern of activation which uniquely identifies its concept. As proposed in Lange and Dyer [19], distributed signatures would be propagated for inferencing over paths of binding *banks* in exactly the same way as scalar signatures. Figure 14 shows an example of this. Similar concepts would have similar distributed patterns of activation as their signatures (or *reduced descriptions*), so that each signature would carry at least a limited amount of semantic content. A first pass at this might entail the use of microfeature-like patterns, as in the distributed model of McClelland and Kawamoto [23], but it would be preferable to have the

[5]The normal coding capacity of connectionist elements is usually in the range of 1-5 bits.

signature patterns learned over time, as done by the model of Miikkulainen and Dyer [25].

One of the most important results of using distributed signatures would be a simplification of the network structure calculating whether individual signature bindings match a role's selectional restrictions (or logical binding constraints). If signatures are distributed patterns of activation that are similar for similar concepts, then selectional restrictions could be computed with a bank of nodes that does a simple *similarity threshold* between the signature binding and the distributed signature of the binding constraint to determine whether signatures should be passed through. The most intriguing possibility, however, is that the binding constraint nodes could be replaced by small distributed ensemble of nodes trained by backpropagation or some other distributed learning mechanism to recognize the signatures that their roles can accept.

We will be exploring distributed signatures and their possible uses for learning binding constraints in the future, along with other ways of using learning based on the semantic content of signatures themselves after the network has performed inferencing. These possibilties for applying learning techniques with distributed signatures make for potentially the most important difference between signatures and other parallel inferencing techniques in which the bindings themselves convey no semantic information, such as marker-passing models and Shastri and Ajjanagadde's [33] synchronization approach.

5.4 EXTENSIONS TO REMIND

The above advances to ROBIN's dynamic representation and inferencing abilities should significantly increase the types and lengths of the stories that ROBIN will be able to disambiguate and understand. Because REMIND is based on ROBIN, these advances should also form the basis for significantly increasing the ability of REMIND to understand and retrieve longer and more complex stories. We therefore plan to extend REMIND to include the advances developed in ROBIN's inferencing abilities. We will then run a number of different simulations to test these new retrieval abilities and how REMIND compares to other models of analogical and case-based retrieval.

Another set of simulations to be run will test how the integration of understanding and retrieval within the single spreading-activation mechanism of REMIND has an effect on the language understanding and disambiguation process. Initial experiments has shown that priming from analogous episodes (cases) activated as part of the retrieval process has a desirable effect on the understanding process by influencing the context in which disambiguation and interpretation of input cues takes place. Furthermore, it appears that many of the desirable features of the explicit indexing methods of case-based reasoning systems emerge from the dynamics of REMIND's spreading-activation process and how episodes are learned over time. For example, one important feature of a useful index is how unique it is. Although REMIND indexes its episodes under all of their features, relatively unique features affect retrieval more than common ones simply because they activate fewer episodes (compare *Possess-Obj* to the more abstract *Avoid-Detection* and *Punishment* frames in Figure 7). Another important aspect of the spreading-activation process is that particularly salient features receive the most activation and therefore automatically act as stronger retrieval indices. We plan to run a number of simulations to explore whether these initial results scale up and whether they can show that the benefits of explicit indexing in CBR models can fall out of the comprehension process. The effect of retrieval of analogous cases on the interpretation process may also show another way that learning can occur in the network.

6 SUMMARY

Our initial research on developing structured connectionist models for natural language understanding and episodic memory retrieval is quite promising. The use of signatures in ROBIN allows it to perform some of the variable binding and parallel dynamic inferencing difficult for connectionist models, while its integration within the constraint satisfaction abilities of its evidential layer allows it to perform disambiguation and reinterpretation difficult for traditional symbolic models. Once a connectionist model can perform some of inferencing and disambiguation of the natural language understanding process, it is a natural extension to have the resulting interpretations directly influence memory retrieval, as appears to be the case in people. REMIND's integration of ROBIN's language understanding networks with sub-networks performing memory retrieval has illustrated many of the potential benefits of this approach. These include several computational benefits over retrieval-only models and the fact that it can potentially account for many psychological phenomena involving priming and language effects on human memory retrieval.

We are currently developing a number of extensions to the inferencing and representational abilities of ROBIN and REMIND's network structure. They include: (1) the ability to handle multiple dynamic instances of each frame, (2) the ability to handle recursive inferences and rules with function terms, (3) the ability to handle more complex rules having conjunctive terms, and (4) exploring the use of representing signatures as uniquely-identifying distributed patterns of activation carrying limited semantic content (i.e. as reduced descriptions) for learning. We plan to integrate these new inferencing abilities into ROBIN and REMIND, allowing us to perform a number of experiments on how they allow the models to scale up and how they compare to current connectionist models of language understanding and to current models of analogical and case-based retrieval. These developments should allow a significant increase in the abilities of connectionist models' reasoning abilities and our understanding of the processes of language understanding and episodic memory retrieval.

ACKNOWLEDGEMENTS

I would like to thank Michael Dyer, Ken Forbus, Keith Holyoak, and Charles Wharton for helpful discussions on the models. I would also like to thank Lawrence Bookman and Ron Sun for the opportunity to present this chapter and for several useful comments and suggestions.

REFERENCES

[1] Anderson, J. R. (1983). *The architecture of cognition.* Cambridge, MA: Harvard University Press.

[2] Barnden, J. (1990). The power of some unusual connectionist data-structuring techniques. In J. Barnden and J. Pollack (eds.), *Advances in Connectionist and Neural Computation Theory.* Norwood, NJ: Ablex.

[3] Barnden, J. and Srinivas, K. (1992). Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. *International Journal of Man-Machine Studies*, 36:221–246.

[4] Bookman, L.A. (1994). *Trajectories through Knowledge Space: A Dynamic Framework for Machine Comprehension.* Boston, MA: Kluwer.

[5] Charniak, E. (1986). A neat theory of marker passing. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, Philadelphia, PA.

[6] Cottrell, G. and Small, S. (1982). A connectionist scheme for modeling word-sense disambiguation. *Cognition and Brain Theory*, 6:89–120

[7] Diederich, J. (1990). Steps toward knowledge-intensive connectionist learning. In J. Barnden and J. Pollack (eds.), *Advances in Connectionist and Neural Computation Theory*. Norwood, NJ: Ablex.

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

[9] Dyer, M. (1983). *In-depth Understanding: A Computer Model of Integrated Processing for Comprehension*. Cambridge, MA: MIT Press.

[10] Gentner, D. and Forbus, K. (1991). MAC/FAC: A model of similarity-based retrieval. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society* (pp. 504-509). Hillsdale, NJ: Lawrence Erlbaum.

[11] Granger, R. H., Eiselt, K. P., and Holbrook, J. K. (1986). Parsing with parallelism: A spreading activation model of inference processing during text understanding. In J. Kolodner and C. Riesbeck (eds.), *Experience, Memory, and Reasoning* (pp. 227-246). Hillsdale, NJ: Lawrence Erlbaum.

[12] Hammond, K. (1989) *Case-based Planning*. Boston: Academic Press.

[13] Hendler, J. (1989). Marker-passing over microfeatures: Towards a hybrid symbolic/connectionist model. *Cognitive Science*, 13:79–106.

[14] Hofstadter, D. and Mitchell, M. (in press). The copycat project: A model of mental fluidity and analogy-making. To appear in J. Barnden and K. Holyoak (eds.), *Advances in Connectionist and Neural Computation Theory, volume II: Analogical Connections*. Norwood, NJ: Ablex.

[15] Holldobler, S. (1990). A structured connectionist unification algorithm. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Boston, MA.

[16] Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review*, 95:163–182.

[17] Kitano, H., Tomabechi, H. and Levin, L. (1989). Ambiguity resolution in DMTrans Plus. In *Proceedings of the Fourth Conference of the European Chapter of the Association of Computational Linguistics.* New York, NY: Manchester University Press.

[18] Kolodner, J., Simpson, R., and Sycara, K. (1985). A process model of case-based reasoning in problem solving. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, p. 284-290. Los Altos, CA: Morgan Kaufman.

[19] Lange, T. and Dyer, M. (1989). High-level inferencing in a connectionist network. *Connection Science*, 1(2):181-217.

[20] Lange, T. (1992). Lexical and pragmatic disambiguation and reinterpretation in connectionist networks. *Interational Journal of Man-Machine Studies*, 36:191-220.

[21] Lange, T. and Wharton, C. (in press). REMIND: Retrieval from episodic memory by inferencing and disambiguation. In J. Barnden and K. Holyoak (eds.), *Advances in Connectionist and Neural Computation Theory, Volume 3: Metaphor and Reminding.* Norwood, NJ: Ablex.

[22] Lytinen, S. (1984). *The organization of knowledge in a multi-lingual integrated parser.* Ph.D. thesis, Research Report 340, Yale University, Department of Computer Science, New Haven, CT.

[23] McClelland, J. L. and Kawamoto, A. H. (1986): Mechanisms of sentence processing: Assigning roles to constituents of sentences. In McClelland and Rumelhart (eds.), *Parallel Distributed Processing: Vol. 2*, p. 272-325. Cambridge, MA: The MIT Press.

[24] Miikkulainen, R. (1993). *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory.* Cambridge: MIT Press.

[25] Miikkulainen, R. and Dyer, M. (1991). Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15:343-399.

[26] Norvig, P. (1989). Marker passing as a weak method for text inferencing. *Cognitive Science*, 13:569-620.

[27] Riesbeck, C. K. and Schank, R. (1989). *Inside Case-based Reasoning.* Hillsdale, NJ: Lawrence Erlbaum.

[28] Riesbeck, C. K. and Martin, C. E. (1986). Direct memory access parsing. In J. Kolodner and C. Riesbeck (eds.), *Experience, Memory, and Reasoning*, pp. 209-226. Hillsdale, NJ: Lawrence Erlbaum.

[29] Rumelhart, D. E., Hinton, G. E., and McClelland, J. L. (1986): A general framework for parallel distributed processing. In Rumelhart and McClelland (eds.), *Parallel Distributed Processing: Vol. 1*, p. 45-76. Cambridge, MA: The MIT Press.

[30] Schank, R. (1982). *Dynamic memory*. NY: Cambridge University Press.

[31] Schank, R. and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding*. Hillsdale, NJ: Lawrence Erlbaum.

[32] Schank, R., and Leake, D. B. (1989). Creativity and learning in a case-based explainer. *Artificial Intelligence*, 40:353-385.

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

[34] St. John, M. (1992). The story gestalt: A model of knowledge-intensive processes in text comprehension. *Cognitive Science* 16:271-306.

[35] Sun, R. (1993). *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. New York: John Wiley and Sons, Inc.

[36] Thagard, P., Holyoak, K. J., Nelson, G., and Gochfeld, D. (1990). Analog retrieval by constraint satisfaction. *Artificial Intelligence*, 46:259-310.

[37] Touretzky, D. (1990). Connectionism and compositional semantics. In J. Barnden and J. Pollack (eds.), *Advances in Connectionist and Neural Computation Theory*, Norwood, NJ: Ablex.

[38] Touretzky, D. and Hinton, G. (1988). A distributed connectionist production system. *Cognitive Science*, 12:423-466.

[39] Waltz, D. and Pollack, J. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation,. *Cognitive Science*, 9:51-74.

[40] Wilensky, R. (1983). *Planning and Understanding*. Reading, MA: Addison-Wesley.

[41] Wharton, C. M. and Lange, T. (1993). Case-Based Retrieval and Priming: Empirical Evidence for Integrated Models. In *Proceedings of the IJCAI-93 Workshop on Reuse of Designs: An Interdisciplinary Cognitive Approach.* Chambery, France, August 1993.

4

Hierarchical Architectures for Reasoning

R.C. LACHER AND K.D. NGUYEN

Department of Computer Science
Florida State University
Tallahassee, FL 32306-4019

1 INTRODUCTION

This chapter has a threefold purpose: (1) to introduce a general framework for parallel/distributed computation, the *computational network*; (2) to expose in detail a symbolic example of a computational network, related to expert systems, called an *expert network*; and (3) to describe and investigate how an expert network can be realized as a neural network possessing a *hierarchical symbolic/sub-symbolic architectural organization.*

A computational network is essentially a directed graph in which each component (vertex or directed edge) has data processing functionality, further endowed with a concept of global network computation. Examples of computational entities that admit descriptions within the computational network model include biological neural networks, artificial neural networks, the parallel virtual machine model of loosely coupled MIMD computation, human collaborations such as committees, and expert networks. Many of the principles of neural network learning can be lifted to the level of computational networks. We present a re-examination of backpropagation learning in this context and derive the computational network backpropagation, or CNBP, learning algorithm.

An expert network is a computational network that can be obtained from an expert system. The architecture of the expert network is derived from the expert system: the network topology from the rule base, the local processing functionality of the vertices and edges from the system of inference, and the global computation concepts from the inference engine. The process of constructing an expert network from an expert system is reversible.

Expert network backpropagation, or ENBP, is a learning method for expert networks obtained as an instantiation of CNBP. ENBP has proven to be useful in knowledge refinement, allowing an expert system builder to make the transition from coarse knowledge, in the form of rough-draft rules, to fine knowledge, in the form of rules with subtlety represented by analog parameters such as certainty factors, using supervised learning and the historical record of expert behavior as a training set.

The symbolic-level nodes of an expert network can be represented by neural networks, which we view as computational networks of sub-symbolic processors. We investigate the optimal architectures for these representations, which provide a realization of an expert network as a neural network with a hierarchical topological organization: a sparsely interconnected collection of densely intraconnected neural nets. This hierarchical sparse/dense organization is analogous to biological neural organization. It captures two levels of knowledge: domain knowledge in the sparse superstructure and metaknowledge in the dense substructures. The hierarchical structural parameters are well within the connectivity constraints found in biology, making feasible the scaling up of neural-based expert networks to sizes comparable to those of living systems.

2 COMPUTATIONAL NETWORKS: A GENERAL SETTING FOR DISTRIBUTED COMPUTATIONS

A computational network is a general framework for parallel/distributed computation modeled on a directed graph in which the vertices and directed edges have computational functionality and for which there is some holistic notion of cooperative computation [32, 34]. Computational paradigms that fit within the computational network framework include biological neural networks; artificial neural networks; distributed computation on a loosely coupled collection of von-neuman machines connected to a digital communications network, as exemplified by Parallel Virtual Machine [54, 55]; human collaborative decision-making and problem-solving; and expert networks [31]. We return briefly to each of these examples after introducing computational network concepts.

2.1 DEFINITIONS AND NOTATION

A *computational network* (CN) is a directed graph together with certain attributes and specifications. These may be *local* or *global*, the former referring to individual CN components (vertices or edges) and the latter to the CN itself. In particular, to compute with a CN one must specify the types of data allowed for (various parts of) the computation; the local functionalities associated with digraph components; a method of timekeeping or scheduling to keep the global computation organized; a method of aggregating the local computations into a global network computation; and how data is to be presented to, and retrieved from, the CN.

Data Types

The types of data with which the computational network is competent must be specified. Depending on the setting, allowed data types may be specific molecules, discrete or continuous numerical values, character data, or even sounds that represent either analog data or discrete symbolic information. Different components of the CN may require different data types, and the CN may operate internally with data types distinct from the I/O data types.

Local Functionalities

The components of a computational network must have computational attributes. Thus each vertex of a CN must have an associated ability to receive data at its incoming edges, process that data into an internal state, compute an output value, and make this output value available to each of its outgoing edges. Each directed edge must have an associated ability to receive data at its initial end, compute a value, and make this value available at its terminal end. We use the terms *node* and *connection* to mean, respectively, a vertex or a directed edge in a CN together with its associated functionality.

Node functionality is broken down into two stages, an input or combining stage and an output or firing stage. In the combining stage a node computes an internal state y from its input data values $x_1, \ldots, x_n$. We denote the function so implied by Γ and call it the *combining function* of the node (or associated with the vertex). After computing its internal state y, a node must compute an output value z. We denote this second function by φ and call it the *firing function* of the node.

The node combining functions of a computational network may be specified in a number of ways, either explicitly or implicitly. For example, if time is continuous Γ may be determined implicitly by a differential equation, whereas if time is discrete Γ may be given by an explicit formula.

Connection functionality transforms the data received by a connection into a transmitted signal value. The input to a connection is the output value z of the node at its initial end. The connection computes one of the input values x for the node at its terminal end. We denote the function making this computation by σ and call it the *synaptic function* of the connection.

Commonly encountered synaptic functions may be linear functions; linear threshold functions; sigmoidal functions; or simple conduits that transmit data unchanged except for a possible time delay.

Time

A notion of timekeeping or scheduling of the various component computations and data transactions is required in order to give meaning to whole-network activation and computation. The possibilities for timekeeping range intrinsic such as self-organization to extrinsic such as management by outside expertise.

Global Activation

The local components of a CN are activated by simply applying their functionality to whatever input they have at any given time. For global computation, however, these local activations must be orchestrated in some way to define a notion of global or network activation. Options include: *synchronous activation*, in which each network component is activated simultaneously; *asynchronous activation*, in which network components are activated randomly one at a time; *event-driven* activation, in which a network component activates when one of its input values changes; and *managed* activation, in which components activate on the command of a central scheduler or manager.

Network Computation

A computational network is intended to be used as a computer, and like a traditional computer the computations of the various parts must be orchestrated into a holistic whole-network computation in some way. In all cases, the global network computation is obtained by successive global activations. The cases differ in how they determine when a network computation is completed. There are two basic choices: either activate for a certain length of time, or activate until the network has reached some kind of global equilibrium state.

Input and Output

A method must be prescribed whereby data values may be introduced into, and retrieved from, a CN from outside the network. For the purposes of this work we will assume that I/O is accomplished by specification of two subsets of nodes (possibly overlapping), "input" nodes and "output" nodes. Data is inserted into the CN to begin a computation by externally setting the states y of the input nodes to the input data values. After network computation is completed, data is retrieved from the CN by reading the outputs z of the output nodes.

2.2 ACTIVATION DYNAMICS

The attributes which collectively define a computational network are not independent. For example, the method of keeping time, the concept of holistic computation, and the meaning of I/O are all interelated, and some choices in one direction may preclude a possibility in another direction. A coherent set of attributes for timekeeping, global activation, network computation, and I/O together constitute the rules for *activation dynamics* of the CN. We consider briefly some of the most often used network computation strategies.

Centrally Managed Computation

Usually used with discrete time, although possible with continuous time. A central entity, such as an operating system or a manager, makes decisions as to timing of local computations and routing of data. Output is read at a time specified by the computation manager.

Synchronous Equilibrium Computation

Used with either discrete or continuous time. This method activates the local functionalities indefinitely at each clock tic (or continuously) until a dynamic equilibrium is reached. Output is read at equilibrium. Classically this equilibrium is assumed to be a fixed point in the space of node states, but more general attractors are sometimes allowed [21, 10, 41, 51]. It may be quite challenging to decide whether appropriate equilibria are always attained in a given CN [19]. Virtually all continuous-time CNs use synchronous activation, and most use equilibrium dynamics to define network computation.

Fixed Time Delay Computation

Used with either discrete or continuous time. The network is activated as in synchronous activation above, but for a certain number of iterations (or length of time) after which output is retrieved. This is often used in place of an equilibrium rule. The time of activation is chosen so that the network will be close to equilibrium upon completion.

Asynchronous Equilibrium Computation

This makes sense only for discrete time. There are two variations, a global one in which a node is chosen at random and its incoming connections and the node itself are activated, and local one in which each component chooses to activate at random times. In either case the process continues until equilibrium is reached. When the probability of local activation is kept small, these produce equivalent equilibrium dynamics [18, 20].

Event Driven Computation

Again for discrete time only. Each component of the CN activates whenever it receives a new input value, until no values change. This is equivalent to synchronous equilibrium dynamics [32]. Expert networks and human collaborations typically use event-driven activation, and results are generally useful only when an equilibrium state is achieved.

2.3 EXAMPLES

In a biological neural network (BNN), the local functionalities are determined by the extraordinarily complex biochemical processes of synaptic transmission, membrane channels, and internal cell chemistry. The synaptic functions reflect the type and density of transmitter molecules, together with properties of the inbound membrane channels of the receiving cell. The combining and firing functions reflect the internal cell biochemical accrual process and the sensitivity and other properties of the outbound membrane channels, respectively.

An artificial neural network (NN) is a mathematical analogy of a BNN. The synaptic and firing functions are usually specified explicitly. Quite typically, the synapses are simple linear functions. The firing functions may be of virtually any type, but most often are sigmoidals such as logistic or hyperbolic tangents, threshold functions (with discrete output), symmetric distributions such as the gaussian, or some combination of these types. In discrete time NNs, the combining function is usually given explicitly, with simple additive accrual being the most common, while in continuous time NNs Γ is more often given implicitly by constraints on its derivatives.

A typical use of parallel virtual machine (PVM) is to perform a computation by parceling out identifiable sub-computations to various computers on a high speed communications network. The synaptic functions are pure transmissions of data, with some small time delay. The node functionalities are quite complex and determined by user programs. Global activation is event-driven and network computation is centrally managed.

In human collaborations (HC) the synaptic functions transmitting human-human communication are again simple conduits, albeit of very complex data. The combining functions reflect the receiving and interpretation by one person of the information supplied by all the others in the collaboration. The firing functions reflect the formulation and transmission of personal information and conclusions out to other members of the collaboration. Activation dynamics can be a form of managed computation, for example when there is a strong leader such as a teacher or supervisor. Often more effective is the committee model, with event-driven activation. There is no guarantee of convergence; convergence is a goal of the collaboration.

An *expert network* (EN) is a computational network derived from a rule-based expert system (production system). The digraph topology is determined by the domain rule base; the local functionalities are determined by the inference system; and the timing mechanism is derived from the computational scheduling method of the expert system shell. ENs typically use discrete time, have acyclic topology, and process analog data. Expert networks are discussed further in following sections.

Table 1 Classification of example CNs.

	BNN	NN	PVM	HC	EN
DD/AD	1	x	0	x	1
DT/CT	1	x	0	0	0
AT/RT	1	x	1	1	0

2.4 CN Classification

There are three broad dichotomies that occur very naturally in the specification of a CN. The five examples discussed briefly above give evidence that the resulting categories are non-vacuous and interesting. These dichotomies, and some notation we will use for the resulting classification, are as follows:

- DD/AD: Discrete or Analog Data
- DT/CT: Discrete or Continuous Time
- AT/RT: Acyclic or Recurrent Topology

Only when compactness of notation is convenient, we use a 3-digit binary encoding to represent a set of choices in these three dichotomies, the left digit representing data type, the middle digit representing time type, and the right digit representing topology type. We also use 'x' as a don't-care or union of types. For example, A CN of type 101 computes with analog data using discrete time and a recurrent topology, while type 10x has the same data and time restrictions but does not specify whether the topology is acyclic or not.

Classification of the five examples discussed above is given in Table 1. Some of the classification choices are arguable, but most will agree that these choices indicate a legitimate point of view within which the paradigm may be studied and that some choice must be made in order to focus the study.

2.5 Discrete Time Computational Networks: Notation

We establish some notation for updating discrete time (or "type x0x") CNs. Similar notation is appropriate for continuous time CNs except that often some of the local functionalities are specified implicitly through differential equations.

A CN consists of nodes and connections organized into a directed graph structure. We will use an adjacency matrix notation system for the CN components based on a labeling of the nodes: a single subscript indicates an association with the vertex so labeled, and a double subscript indicates association with a directed edge, with "assignment statement order" for the edge subscripts: a subscript ji indicates association with the edge from vertex i to vertex j. In this notation, Γ_j and φ_j are the combining and firing function, respectively, of node j, and σ_{ji} is the synaptic function of the connection from node i to node j. We also use z to denote node output (or activation value) and y to denote node internal state. If the ji synapse is linear, then $\sigma_{ji}(z_i) = w_{ji}z_i$, where w_{ji} is the weight of the connection. We assume in this discussion that the node labels constitute an enumeration $1, \ldots, n$.

The *internal state* of the CN at a particular time t consists of all the node states $y_j(t)$ usually collected into a vector $y(t) = (y_1(t), \ldots, y_n(t))$. Similarly, the *activation state* of the CN at time t is the vector $z(t) = (z_1(t), \ldots, z_n(t))$ of local activation values at time t. (It should be kept in mind that important properties of these state vectors are symmetric, that is, independent of the particular vertex ordering.) New states are calculated using the *update equations*

$$
\begin{aligned}
x_{ji} &:= \sigma_{ji}(z_i) \;\; for \;\; i = 1, \ldots, n && (4.1a)\\
y_j &:= \Gamma_j(x_{j1}, \ldots, x_{jn}) && (4.1b)\\
z_j &:= \varphi_j(y_j) && (4.1c)
\end{aligned}
$$

during three time steps (or in one time step split into three sub-steps). How local updates are organized into network activation varies as discussed earlier. Activation dynamics is the study of the behavior of network states as they change over time.

One requirement not often made explicit for computational networks is *symmetry* of combining functions: Γ_j should give output that is independent of the labeling order of the nodes. Another system of notation that makes this requirement more obvious is based on predecessor/successor relations in the network topology. Define a *predecessor* of node j to be any node in the network that initiates a connection into j. The set of predecessors of j is defined and denoted as

$$Pred(j) = \{i | there\ is\ an\ edge\ from\ i\ to\ j\}.$$

The update equations can be restated in terms of predecessors as follows. First compute post-synaptic input for node j:

$$x_{ji} := \sigma_{ji}(z_i)$$

for all $i \in \mathit{Pred}(j)$, where σ_{ji} is the synaptic function of the connection from i to j. Denote the vector of all post-synaptic input for node j by x_j. This vector has dimension $|\mathit{Pred}(j)|$, one component x_{ji} for each $i \in \mathit{Pred}(j)$, but the order of components is not important. Next compute the internal state of node j:

$$y_j := \Gamma_j(x_j)$$

where Γ_j is the combining function for node j. Γ_j is a symmetric function of $|\mathit{Pred}(j)|$ variables. Finally compute the activation value of node j:

$$z_j := \varphi_j(y_j)$$

where φ_j is the output function of node j.

We will generally stick to the simpler adjacency matrix notation of equations 4.1. This simplicity does blur certain subtleties, however, by making the tacit assumption that computation doesn't need to distinguish between no connection from i to j and a connection from i to j with $\sigma_{ji} \equiv 0$. Cases can arise where this distinction is important. In such cases the missing connectivity information can be maintained in a seperate adjacency matrix. If the network is sparse, it may be appropriate to use more compact representations such as adjacency lists that implement the predecessor/successor notation.

3 TYPE x00 COMPUTATIONAL NETWORKS

For the remainder of this chapter we restrict our attention to computational networks of type x00, that is, we assume discrete time and acyclic topology but allow either discrete or analog data types both internally and as I/O.

3.1 ACTIVATION DYNAMICS

Activation dynamics of acyclic, discrete-time computational networks may assume any of the forms discussed in Section 2.2. Synchronous, asynchronous, and event-driven activation are all equivalent to fixed time delay (if the delay is appropriately large) and all result in reaching a terminal activation state in finite time [32]. In other words, given a CN of type x00, we can activate using any of these methods for a globally fixed amount of time, after which activation will cease to produce changes in any of the internal states of the CN. This activation may be component-parallel, component-distributed, or component-serial. The type x00 CN thus becomes a (parallel/distributed) computer with a fixed number of local computations required for I/O: insert input, compute a fixed number of local activations, then retrieve output. The local computations consist of applying the update assignment statements given by equations 4.1 until a steady activation state $(z_1, \ldots, z_n)$ is reached. We call this the *terminal activation state* of the network and refer to the component z_j as the terminal activation value of node j.

3.2 Influence and Error

Backpropagation is one of the most widely known and successfully used connectionist learning methods [57, 44]. Most often, backpropagation is applied to layered feedforward computational networks with the kind of simple processing functionality associated with low-level, sub-symbolic networks: linear synapses, additive combining functions, and sigmoidal or gaussian output functions. Many of the ingredients of backpropagation learning can be generalized for general computational networks. For CNs, the standard algorithm requires two changes: localize forward and backward activation to free the algorithm of the layer structure, and decouple the process of node error assignment from the weight correction step. The first is described previously and in [36]. The second uses the concept of influence factor, introduced in [32]. Influence factors are associated with connections and specific network input. The *influence factor* ε_{kj} of the connection from node j to node k is the rate of change of output of node k with respect to the output of node j, evaluated at the terminal activation state: $\varepsilon_{kj} = \partial z_k / \partial z_j (z_j)$. Expanding this derivative using the chain rule we obtain

$$\varepsilon_{kj} = \varphi_k'(y_k) \times \frac{\partial \Gamma_k}{\partial x_{kj}}(x_{k1}, \ldots, x_{kn}) \times \sigma_{kj}'(z_j). \tag{4.2}$$

Again we emphasize that influence factors are dependent on particular network input: The derivative of φ_k is evaluated at the terminal internal state of node k, the partial of Γ_k is evaluated at the terminal post-synaptic input to node k, and the derivative of σ_{kj} is evaluated at the terminal output of node j. Influence factors are associated with connections and are calculated during forward activation of the network.

Once influence factors have been calculated for all the connections of an acyclic CN during forward activation, error can be assigned to all of the nodes in the CN during a reverse activation. This reverse activation is in essence an activation of the reverse of the CN. The reverse network topology consists of the vertices and edges of the original network, but with all edge orientations reversed. The nodes of the reverse network use summation combining function and the identity activation function, i.e., the nodes are linear units. The synaptic functions are also linear with weight equal to the influence factor. Note that the reverse network is also acyclic.

Error is assigned to each of the output nodes using equation 4.3a, where I is ideal output, and to all non-output nodes using equation 4.3b:

$$e_j := I_j - \zeta_j, \tag{4.3a}$$
$$e_j := \sum_k \varepsilon_{kj} e_k. \tag{4.3b}$$

Applying equation 4.3b recursively is an activation of the reverse network with input given by 4.3a. The resulting terminal reverse activation state is an error assignment throughout the network. The error assignment process works in any acyclic CN for which the derivatives of equation 4.2 are defined.

3.3 LOCAL GRADIENT DESCENT

Once error has been distributed among the nodes in a computational network, we can apply gradient descent learning both *selectively* and *locally* to any node whose incoming synapses are linear. This decoupling of error assignment and learning means we can allow more complex synaptic functionality into certain nodes, we can have hard-wired connections into perhaps other selected nodes, and suppress learning at any selection of sites, while maintaining a global learning process. Local gradient descent amounts to applying the Widrow-Hoff delta rule using local error.

Calculating the gradient of squared local error at node j with respect to synaptic weights $w_{j1}, \ldots, w_{jn}$ and taking a step in the opposite direction yields the following learning rule:

$$\Delta w_{ji} = \eta e_j \varphi_j'(y_j) \frac{\partial \Gamma_j}{\partial x_{ji}}(x_{j1}, \ldots, x_{jn}) z_i + \mu \Delta w_{ji}^{prev}. \tag{4.4}$$

This equation defines one learning step with *learning rate* η and *momentum* μ in the direction of steepest descent of square local error.

3.4 CNBP

Putting all these components together results in a learning method called Computational Network BackPropagation, or CNBP. CNBP applies to type x00 CN at any nodes with linear incoming synapses. All that is required to complete an implementation of CNBP is calculation of the derivatives appearing in equations 4.2 and 4.4. CNBP is summarized as follows.

Assume given a set of training exemplars (ξ^l, I^l), $l = 1, 2, \ldots$ consisting of input $\xi^l = (\xi^l_1, \ldots, \xi^l_m)$ and ideal output $I^l = (I^l_1, \ldots, I^l_n)$. The basic learning process goes as follows:

```
Initialize
    present $\xi^l$ to input nodes
Activate
    calculate terminal activation state $z^l_j$ for each node
    calculate influence factors $\varepsilon_{kj}$ for each connection
Initialize error
    present external error $e^l_k = I^l_k - z^l_k$ to output nodes
Reverse activate
    activate the reverse network, assigning error $e^l_j$ to each node
Learn
    change soft weights using local gradient descent
```

The learn step can be carried out after each exemplar presentation (on-line learning) or accumulated and carried out at the end of an epoch (batch learning). The entire procedure loops until error is reduced sufficiently.

4 Expert Systems

An expert system (ES) captures domain-specific knowledge and uses this knowledge to reason about problems in the domain. By far the most successful type of expert system so far has been the rule-based system [15]. A rule-based expert system consists of an inference engine that defines and executes the rules of inference and a rule base that comprises the domain-specific knowledge of the system.

Rule-based expert systems have become a mature advanced technology, with many successful software shells on the market, whether success is measured by technical achievement or commercial viability. Three of these are particularly pertinent to the research, development, and production discussed here: M-4[1], CLIPS[2], and G2[3]. These products are all in significant use in a wide variety of application domains by a heterogeneous user community. Commercial users of M-4 have valuable (and proprietary) rule bases ranging in size from a few dozen to ten thousand rules.[4] CLIPS has an avid following despite its lack of user amenities, due in part to its low cost. G2 is the most elaborate (and costly) of the three, with commercial site licenses listing at $42,000. Many of the worlds largest corporations have signed with Gensym for developing their real-time expert system needs, including ASEA Brown Boveri, GE, Monsanto, Occidental Petroleum, Boeing, Du Pont, Texaco, Lafarge Coppee, and 3M [17].

These three shells each deal with uncertainty using a form of EMYCIN logical semantics. M.4 is discussed in some detail below; Hruska and coworkers have constructed a superset of CLIPS that uses essentially the same uncertainty semantics as M.4 [45]; and G2 uses a classical version of fuzzy inference.

4.1 EMYCIN

A seminal demonstration of the efficacy of rule-based systems was a medical diagnosis and treatment advisory system for infectious diseases called MYCIN[48, 1]. A natural consequence of the success of MYCIN was its abstraction to an "expert system shell" in order to apply the same reasoning automation in other domains. A shell is just an expert system with an empty knowledge base and a user interface system to facilitate the insertion and modification of rules. A shell that implements the MYCIN reasoning system is called EMYCIN (for "Empty MYCIN"). M.4 is a commercially available EMYCIN shell. The computational experiments discussed below are based on M.4. The features of EMYCIN inferencing that are important in what follows are the evidence accumulator and the various logical operations [49, 14].

A rule in EMYCIN has the form

$$IF\ a\ THEN\ b\ (cf)$$

where a and b are assertions and cf is a *certainty factor* or confidence factor associated with the rule. The certainty factor may take on any value in the

[1]Product of Cimflex Teknowledge Corporation.
[2]Designed and Produced by NASA, distributed as shareware.
[3]Product of Gensym Corporation.
[4]Private communication from representative of Cimflex Teknowledge

range $-1 \le cf \le 1$. We use the notation $cf_{b|a}$ to denote the certainty value of the implication IF a THEN b. Certainty factors are static numerical attributes of rules. They reside in the knowledge base and do not change during inferencing.

An assertion b may take on an *evidence value* (also sometimes called a certainty factor). The evidence value of an assertion is dynamically updated during inferencing, either through assignment when a query is made or through calculation in terms of evidence values of other assertions previously calculated or assigned during the inference. We denote the evidence value of assertion b by y_b. y_b may range in the interval $-1 \le y \le 1$. The dynamically calculated evidence value of an assertion may be interpreted as a degree of confidence or correctness of the assertion. The evidence value y_b is then converted to a *firing value* z_b through the use of a threshold or other postprocessing criterion. The firing value (in this version of EMYCIN) is restricted to the range $0 \le z \le 1$.

Suppose that we have a current dynamic evidence value y_b for assertion b and subsequently encounter another assertion IF a THEN b (cf). Then EMYCIN adjusts y_b by adding an amount proportional to the firing value z_a for a, the certainty factor $cf = cf_{b|a}$ for the rule, and the proximity of y_b to its domain limits. (When the current evidence value y_b and the rule certainty factor $cf_{b|a}$ have opposite signs, a mediation process is used instead.) The output value z_b for assertion b is then updated by applying the firing criterion to y_b. The firing criterion may vary somewhat from one EMYCIN shell to another. M.4 uses the linear-threshold firing function with threshold value of 0.2.

This update process breaks naturally into three steps. First calculate the certainty-mediated input evidence:

$$x_{b|a} := cf_{b|a} \times z_a; \tag{4.5}$$

then update the evidence value:

$$y_b^{new} := \begin{cases} y_b + x_{b|a}(1 - y_b)\,, & \text{if both } y_b \text{ and } x_{b|a} \text{ are positive,} \\ y_b + x_{b|a}(1 + y_b)\,, & \text{if both } y_b \text{ and } x_{b|a} \text{ are negative,} \\ \frac{x_{b|a} + y_b}{1 - min\{|y_b|, |x_{b|a}|\}}\,, & \text{otherwise;} \end{cases} \tag{4.6}$$

then recalculate the firing value:

$$z_b := \begin{cases} y_b\,, & \text{if } y_b \ge 0.2; \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.7}$$

This firing value is then used as input to other rules of the form IF b THEN c (cf), and so on, until all firing values are stabilized. The inference process begins with external setting of the firing values of selected rule antecedents and spreads through the rule base under control of the inference engine. After the inference process terminates, the values of consequents with non-zero values constitute the conclusions of infcrence.

The reader will probably have noticed the similarity between the equations above and equations 4.1 as well as a principal distinction: 4.6 represents an accumulation process over rules with b as consequent, while 4.1b represents the evaluation of a combining function over all incoming connections simultaneously. We give a closed form version of 4.6 in the next section.

EMYCIN shells differ somewhat in their treatment of logical operations, although they typically use minimum and maximum for AND and OR, respectively, and some kind of inversion for NOT. The differences among shells appear in the way these values are thresholded (or otherwise postprocessed), after applying this common calculation, to determine whether the compound assertion fires. Generally, rules are allowed to have compound antecedents (using the defined logical operations) but compound consequents are discouraged.

M.4 recognizes three logical operations explicitly: AND, NOT, and UNK. The UNK (for "unknown") operation is a version of NOR (NOT following OR).[5] For AND, M.4 uses the same firing function as for evidence combining, given above by 4.7. For NOT, M.4 uses a firing function that is a strict threshold, with threshold value 0.8, resulting in discrete values for NOT and NOR operations.

Each of the operations can be described in three functional steps analogous to 4.5, 4.6, and 4.7 above. These operations, along with the evidence accumulation process, provide functionality to the vertices and edges of an inference network model of the knowledge base, resulting in a computational network. We describe this network, along with explicit M.4 functionalities, in detail in the next section.

[5]M.4 does not recognize an explicit OR operation, hence the non-standard terminology. M.4 implicitly uses two different versions of OR – the DeMorgan dual of AND as well as the evidence accumulator.

5 EXPERT NETWORKS

Expert Network learning technology, a process developed by a group at FSU[6] in partnership with the Florida High Technology and Industry Council, provides a means of automated knowledge refinement in rule-based expert systems. In settings where sufficient historical data exists, expert network learning can significantly improve both the development time and the ultimate level of expertise captured in an expert system project.

The expert network method, at the algorithm level, is a method for knowledge refinement in a rule-based expert system that uses uncertainty. The uncertainty theory can be that of EMYCIN certainty factors as in M-4, fuzzy logic as in G2, probability, Dempster-Shaffer theory, or any other theory that uses a continuously variable value or values to define a level or degree of certainty to implications and/or factual statements. In all such systems the role of uncertainty is to represent the subtle variations of knowledge that, once discovered and captured, complete the transition from coarse novice-level knowledge to refined expertise.

Expert networks allow these systems to make this passage from novice to expert through neural network style learning from data rather than from laborious human expert tinkering. The data required may be either historical records of correct inferences, in which case the learning methods are supervised, particularly Expert Network BackPropagation (ENBP); or the data may be in the form of critique of the expert system's conclusions by experts, in which case the learning methods are reinforcement methods such as Expert Network Temporal Difference (ENTD(λ)). The critical technology implementing both of these learning methods is that of influence factors.

The expert network, or ExNet, technology consists of two major components: Translation and Learning.

Translation

The rule base is translated into a directed graph. The vertices of this digraph represent atomic-level factual statements or actions; these are the antecedents and consequents of the rules. The directed edges represent implications.

The logical semantics, or rules of inference, of the expert system, including the rules dealing with uncertainty, are used to assign information processing

[6]Lacher, Hruska, and Kuncicky

functionality to the vertices and edges. Thus the digraph becomes a computational network. This is called the *expert network* associated with the original expert system.

After the expert network has been modified during the learning phase (described below), the modified expert network is translated back into expert system form, resulting in a new, or refined, set of rules that have optimized performance with respect to the training data. This step requires nothing more than applying the inverse of the translation process.

Learning

Neural network learning methods are applied to the expert network. This learning process results in changes in the parameter values for the uncertainties in the rules, optimized for set of correct inference instances data set (i.e., history). There are several difficult problems to overcome to make this idea actually work, including how to assign a local error to the nodes and how to reduce this local error through gradient descent. We have worked out and implemented all details of this idea in the case of EMYCIN (M-4) and for fuzzy inference. The solutions are detailed in the papers [31, 32, 34, 39]. When the expert system uses EMYCIN certainty factors and/or fuzzy logic to capture uncertainty, ExNet has been completely derived, proved, tested, and covered with patents (pending). In the following treatment we restrict to the M.4 instantiation of EMYCIN.

5.1 TRANSLATION

The network topology is constructed in two stages. First an inference network is created from the rule base. Each vertex in this network represents an antecedent or consequent of a rule and each directed edge represents a rule. The certainty factor of the rule is placed on the edge as a weight. Thus a rule of the form

$$IF\ \ a\ \ THEN\ \ b\ \ (cf)$$

where a and b are assertions and $cf \equiv cf_{b|a}$ is the certainty or confidence factor, defines a connection

$$a \xrightarrow{cf} b.$$

At this point we have constructed an inference net in the usual sense (see [15], page 237).

The evidence accumulation process (equations 4.5, 4.6, and 4.7) of the inference engine defines functionality for the vertices of this inference net, and the edges process initial to terminal value by multiplication by cf (defining linear synaptic functions). The resulting computational network is the first order expert network defined by the expert system. Note that all of the nodes in this network represent assertions; they are called *regular* or *evidence* nodes and denoted as REG nodes.

The second stage of construction is to expand each regular node that represents a compound antecedent statement into a subnetwork. A regular node antecedent such as in the connection

$$OP(a_1, \ldots, a_k) \xrightarrow{cf} b$$

expands to the subnetwork

$$\begin{array}{ccc} a_1 & \xrightarrow{1} & OP \\ \cdot & & \cdot \;\; \cdot \\ a_k & \xrightarrow{1} & OP \\ OP & \xrightarrow{cf} & b. \end{array}$$

Those a_i that are consequents of other rules are already represented by existing nodes. New nodes are created for the other a_i. A connection of weight 1 is added from each a_i to the new OP node, and a connection of weight cf added from the OP node to the consequent b replaces the original outgoing connection. All connections into OP nodes have fixed weight 1 and are called *hard* connections. Connections into REG nodes have weight originating as a certainty factor of a rule and are called *soft* connections.

The combining function for an OP node performs the logical computation defined by the rules of inference used by the expert system. The output function for an OP node is the firing condition for the logical operation. The resulting computational network is the second order expert network defined by the expert system.

Note that there are two kinds of nodes in the second order expert network: REG nodes representing assertions and OP nodes representing logical operations. Note also that all synaptic functions are linear with weights as already described above: soft connections (incoming to REG nodes) have weight cf and hard connections (incoming to OP) have weight 1. Thus synaptic functionality is completely specified. We now give more detailed descriptions of the node functionalities in EMYCIN/M.4 expert networks.

REG *Nodes*

The EMYCIN evidence accumulator given by equation 4.6 can be written in closed form. Let b be a REG node, and suppose b has at least one predecessor in the expert network. (In the parlance of expert systems, b is an assertion that is consequent to at least one other assertion.) For each predecessor a of b let $x_{a|b}$ denote the corresponding post-synaptic input $cf_{b|a} \times z_a$.

The positive and negative evidence values for regular node b are given by

$$\begin{aligned} y_b^+ &= +1 - \prod_{x_{b|a}>0} (1 - x_{b|a}) \; and && (4.8a) \\ y_b^- &= -1 + \prod_{x_{b|a}<0} (1 + x_{b|a}), && (4.8b) \end{aligned}$$

respectively. Positive and negative evidence are then reconciled, yielding the internal state of the node as the value of the REG combining function:

$$y_b := \Gamma_{REG}(x_{b|1}, \ldots, x_{b|n}) \equiv \frac{y_b^+ + y_b^-}{1 - min\{y_b^+, -y_b^-\}}. \tag{4.9}$$

Note that Γ_{REG} is a symmetric function. The only input variables which affect the values of Γ_{REG} are those labeled by predecessors of b, and we could use alternative notation (as described in section 2) to reflect this fact. The notation above assumes that $x_{b|a} = 0$ whenever a is not a predecessor of b.

The output function φ_{REG} for a regular node b is the firing function for assertions defined by equation 4.7:

$$z_b := \varphi_{REG}(y_b) \equiv \begin{cases} y_b\,, & \text{if } y_b \geq 0.2; \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.10}$$

OP *Nodes*

Consider an AND node generated by the antecedent of the rule

$$IF \;\; a_1 \; AND \; a_2 \; AND \; \ldots \; AND \; a_k \;\; THEN \;\; b \;\; (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. Let a denote the compound antecedent AND$(a_1, \ldots, a_k)$. Thus a is an OP node in the second order network. To define the logical AND operation as a function of dynamic evidence values is to define the combining and firing functions of a.

The combining function for a is given by

$$y_a := \Gamma_{AND}(x_1, \ldots, x_k) \equiv \min_i\{x_i\} \tag{4.11}$$

where $x_i = z_{a_i}$ is post-synaptic input. The output function is the same threshold function used for REG nodes:

$$z_a := \varphi_{AND}(y_a) \equiv \begin{cases} y_a\,, & \text{if } y_a \geq 0.2; \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.12}$$

A NOT node such as generated by the antecedent of

$$IF\ \ NOT\,a\ \ THEN\ \ b\ \ (cf)$$

has only a single incoming connection, from a. The combining function is given by

$$y := \Gamma_{NOT}(x) \equiv 1 - x \tag{4.13}$$

(where $x = z_a$) and the output function is

$$z := \varphi_{NOT}(y) \equiv \begin{cases} 1\,, & \text{if } y \geq 0.8; \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.14}$$

An UNK node may be generated by the antecedent of a rule such as

$$IF\ \ a_1\,UNK\,a_2\,UNK\ \ldots\ UNK\,a_k\ \ THEN\ \ b\ \ (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. Let a denote the compound antecedent UNK$(a_1, \ldots, a_k)$. The combining function for a is given by

$$y_a := \Gamma_{UNK}(x_1, \ldots, x_k) \equiv 1 - \max_i\{x_i\} \tag{4.15}$$

where $x_i = z_{a_i}$ is post-synaptic input. The output function is the same as for NOT:

$$z_a := \varphi_{UNK}(y_a) \equiv \begin{cases} 1\,, & \text{if } y_a \geq 0.8; \\ 0\,, & \text{otherwise.} \end{cases} \tag{4.16}$$

Notwithstanding the fact that M.1 does not explicitly acknowledge an OR operation, we could define an OR node that might be generated by the antecedent of a rule such as

$$IF\ \ a_1\,OR\,a_2\,OR\ \ldots\ OR\,a_k\ \ THEN\ \ b\ \ (cf)$$

for some assertions (nodes) $a_1, \ldots, a_k$ in the expert net. As usual letting a denote the compound antecedent OR($a_1, \ldots, a_k$), we define the combining function for a to be

$$y_a := \Gamma_{OR}(x_1, \ldots, x_k) \equiv \max_i \{x_i\} \tag{4.17}$$

where $x_i = z_{a_i}$ is post-synaptic input and the output function to be the same as for AND:

$$z_a := \varphi_{OR}(y_a) \equiv \begin{cases} y_a, & \text{if } y_a \geq 0.2; \\ 0, & \text{otherwise.} \end{cases} \tag{4.18}$$

Given this definition of OR, it is easily verified that UNK = NOT(OR).

Logical Functions

Composing appropriate functions given above yields the following throughput functions (from input to firing value) for logical operations in M.4:

$$AND(x_1, \ldots, x_k) := \begin{cases} min\{x_i\}, & \text{if } min\{x_i\} \geq 0.2, \\ 0, & \text{otherwise;} \end{cases} \tag{4.19a}$$

$$NOT(x) := \begin{cases} 1, & \text{if } x \leq 0.2, \\ 0, & \text{otherwise;} \end{cases} \tag{4.19b}$$

$$UNK(x_1, \ldots, x_k) := \begin{cases} 1, & \text{if } max\{x_i\} \leq 0.2, \\ 0, & \text{otherwise;} \end{cases} \tag{4.19c}$$

where as usual x is interpreted as post-synaptic input for the node (or current evidence value during inference).

5.2 LEARNING

An EMYCIN expert network satisfies all the requirements for CNBP learning: an acyclic CN with linear synapses. Learning can take place only at soft connections (connections into regular nodes), but of course all connections must be used in the error assignment process.

Of the derivatives appearing in equations 4.2 and 4.4, σ'_{kj} is just the weight w_{kj} of the kj connection, and φ'_j is easily calculated, but may vary because of choices of φ made during a particular implementation. If we can calculate (or "define"[7]) the partial derivatives of the node combining functions then we can implement CNBP in expert networks.

[7]The CNBP learning algorithm is sufficiently robust to accommodate approximations. Thus if an approximate derivative can be devised it may work as well as a real derivative.

For a REG node k the partial derivatives are given by

$$\frac{\partial \Gamma_{REG}}{\partial x_{kj}}(x_k) = \begin{cases} \frac{1}{1-x_{k|j}} \frac{1-y_k^+}{1+y_k^-}, & \text{if} y_k^+ \geq |y_k^-| \text{ and } x_{kj} > 0; \\ \frac{1}{1-x_{k|j}} \frac{1+y_k^-}{1-y_k^+}, & \text{if } y_k^+ < |y_k^-| \text{ and } x_{kj} > 0; \\ \frac{1}{1+x_{k|j}} \frac{1-y_k^+}{1+y_k^-}, & \text{if } y_k^+ \geq |y_k^-| \text{ and } x_{kj} < 0; \\ \frac{1}{1+x_{k|j}} \frac{1+y_k^-}{1-y_k^+}, & \text{if } y_k^+ < |y_k^-| \text{ and } x_{kj} < 0 \end{cases} \tag{4.20}$$

provided $x_{k|j} \neq \pm 1$. Here $x_{k|j}$ is $cf_{k|j} \times z_j$, the post-synaptic input to node k from node j, and $y_k^{\pm}$ is given by equation 4.8. (See [36] for details.)

For AND nodes we have

$$\frac{\partial \Gamma_{AND}}{\partial x_{kj}} = \begin{cases} 1 & , \text{ if } x_{k|j} = min_i\{x_{k|i}\} \\ 0 & , \text{ otherwise} \end{cases} \tag{4.21}$$

It is interesting to examine what this means for reverse error assignment: the AND node assigns error backward through node k acting as a demultiplexer switch to the line with lowest incoming value.

Similar results hold for NOT and UNK nodes.

5.3 ENBP

Having calculated the derivatives appearing in equations 4.2 and 4.4, we can apply CNBP in the context of expert networks. This instantiation of CNBP is called Expert Network BackPropagation, or ENBP.

ENBP has been tested on several M.4-based expert systems, including the Wine Advisor [9] and the Control Chart Selection Advisor of Dagli and Stacey [3, 23, 24]. A functioning expert system is used to define expert knowledge by generating specific examples of correct reasoning. In *ablation* testing, a set of soft connections is ablated by setting their connections weights to zero. In *refinement* testing, all of the soft connections are initialized to the neutral value 0.5. The object of the tests is to determine whether the network can recover the knowledge embodied in the connection weights.

In these tests, both learning and generalization have worked remarkably well. The algorithms converge the ablated system to a knowledge state that correctly inferences on the training set, and generalization is perfect: the new system reasons correctly on all possible inputs. Moreover, the ratio of training set size to test set size is small. For example, as few as 22 correct inferences are required to move a 25-connection ablation of wine advisor (a 97 node expert network) to a system that inferences correctly on all 6,912 sensible input queries [34]. Refinement tests have yielded 95–100% generalization rates using training sets of 40 or more exemplars [9].

Mahoney and Mooney subsequently (but independently) developed a version of ENBP (which they call "CFBP") [37]. They are also developing a constructive method using ENBP to enhance the network connectivity at the output nodes that shows significant promise when compared to existing methods [38].

6 NEURAL NETWORKS

By a neural network (NN) we mean a discrete-time computational network with linear synapses, linear combining functions, and non-decreasing firing functions. An expert network is a symbolic computer. The individual nodes have externally assigned and understood meaning - either assertion or logical operation - and the dynamically computed and transported values also have external meaning - degrees of certainty in a conclusion. A neural net, in contrast, is a sub-symbolic computer. The individual nodes and values have external meaning only collectively and selectively. In most cases, an individual node in a NN has no identifiable meaning to an outside observer.

It has been argued that an expert network can be realized as a neural network by replacing each node in the EN with a small NN [34]. We present here some results on the practicality of that process.

6.1 OPTIMAL ARCHITECTURES

We are interested in finding the "optimal" sub-symbolic NN to replace a symbolic node in an expert net. We consider here two nodes types: REG and AND. We restrict our investigation to the class of layered feedforward networks with one hidden layer and sigmoidal output functions, and we use standard backpropagation to train these networks. (See [52, 53] for similar considerations.) Thus the only architectural variable is the number of units in the hidden layer. Our working definition of "optimal" is as follows.

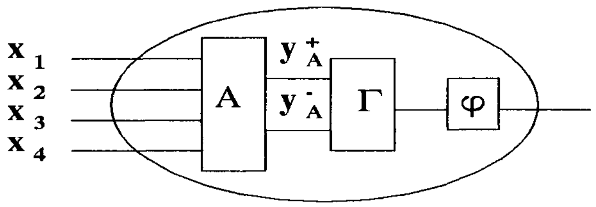


Figure 1 NN for REG node.

For a given architecture we train the NN until generalization error reaches a minimum value. Generally the generalization, or test, error reaches a minimum and begins to increase due to the "overtraining effect". The state of the NN at this minimum generalization error is saved as the acceptable state for that NN. This test is repeated a number of times to obtain an average minimum generalization error (MGE) for a given architecture. The MGE is then plotted as a function of the architecture. As the number of units increases, this plot can be expected to reach a minimum and begin to increase due to the "memorization effect". The architecture that attains this minimum MGE is our optimal architecture .

It is now well known that many functions can be approximated by neural networks (see [12, 22] for example). In particular, all of the node combining and output functions for EMYCIN/M.4, given section 5.1, can be approximated with NNs. We now present some experimental results on finding these approximations. All of the data discussed below was generated using 50 randomly generated training exemplars and 10 randomly generated test exemplars for each training run on each architecture. Five such training runs were made for each architecture, and the average training and testing error over all five runs was used to determine MGE for each architecture.

We are investigating two methods of constructing REG nodes. The first, shown in Figure 1, uses a parallel evidence network (labeled A) followed by a reconciler (Γ). To determine an "optimal" architecture for the evidence network we follow the process described in section 6.1 above. The results of averaging five training trials on a 4-4-1 architecture for a 4-input y^+ network are illustrated in Figure 2. The generalization curve attains a minimum at 41 epochs with $MGE = 5.6 \times 10^{-3}$. The MGE for 4-n-1 architectures, n= $2, \ldots, 9$, are given in Figure 3. These computations show that 4-6-1 is the optimal architecture (in the class under consideration) for the 4-input y^+ network, with $MGE = 3.9 \times 10^{-3}$.

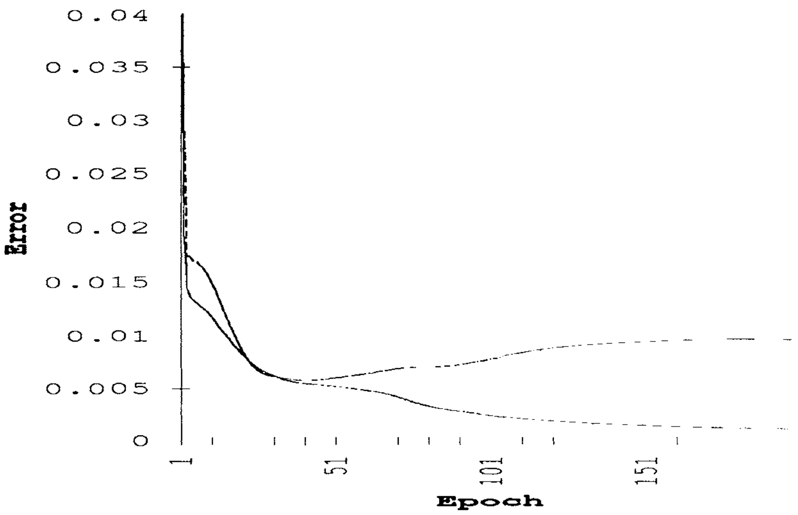


Figure 2 Training and generalization error for 4-4-1 y^+ network (average of five runs).

The second architecture we are currently testing for REG nodes is a modular construction as illustrated in Figure 4. The modularity is based on the accumulation of evidence as given in equation 4.6. Modularity allows us to concentrate on solving the 3-input REG problem and then build more general REG nodes using extant components. The modules labeled A and B in Figure 4 are identical 3-n-2 NNs that take three evidence values as input and give the values y^+ and y^- as output. Once the optimal 3-n-2 module is trained it can be used in cascade fashion to build an evidence accumulator for any number of inputs. The savings in training effort is offset by loss of parallelism in the evidence computation. The MGE plot for 3-n-2 indicates that 3-6-2 is optimal.

We have tested the idea of replacing symbolic nodes with these subsymbolic networks and subsequently training the EN/NN with ENBP (as in section 5.3). For the small expert net we used for testing this experiment worked as one would expect: the EN/NN learned with about the same efficiency as the original EN.

We are carrying out exhaustive experiments to determine optimal architectures for y^+, y^- as well as "black box" REG nodes with n inputs, $n = 3, 4, \ldots$.

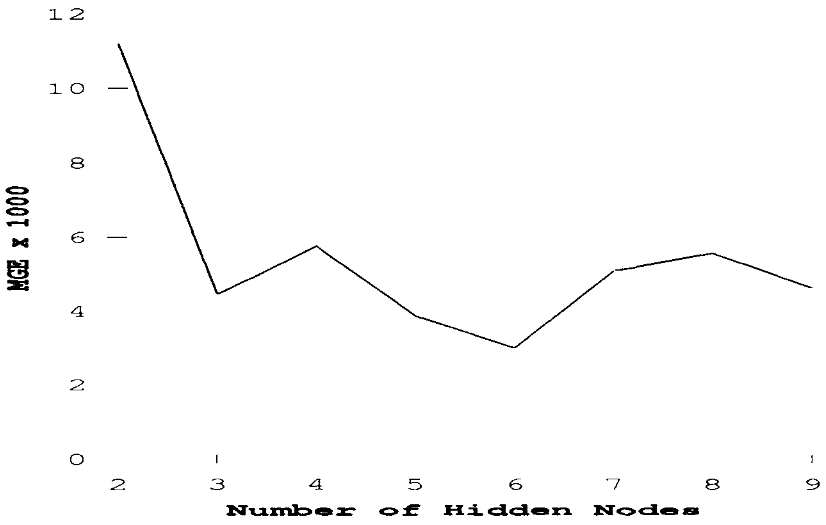


Figure 3 Minimum generalization error $\times 1000$ for 4-n-1 y^+ networks.

These should give a good picture of how parallel REG NNs scale with the number of inputs. For models of human reasoning, however, this scaling may be irrelevant: it seems likely that the modular architecture approach more closely resembles human evidenciary techniques - we tend to weigh evidence a few components at a time and "build a case" rather than process many pieces of evidence in parallel. What even these preliminary results show is that symbolic-level nodes in an expert network can be built with very simple sub-symbolic neural networks and standard training techniques.

7 SUMMARY

We have defined a general framework for parallel/distributed computation, the computational network, or CN. Examples of computational phenomena that admit descriptions within the CN model include biological neural networks, artificial neural networks, the parallel virtual machine model of loosely coupled MIMD computation, human collaborations such as committees, and expert networks. A CN is essentially a directed graph in which each component

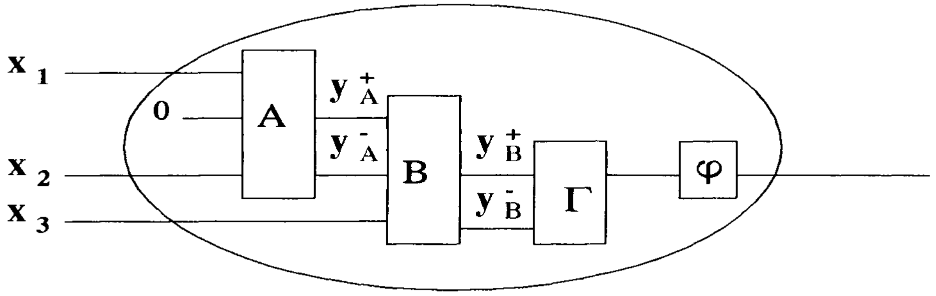


Figure 4 Modular NN for REG node.

(vertex or directed edge) has data processing functionality, further endowed with a concept of global network computation.

A computational network can be classified according to whether it (1) processes discrete or analog data, (2) uses discrete or continuous time, and (3) has an acyclic or recurrent network topology. Expert networks are CNs of "type x00", according to this classification.

The principles of backpropagation learning are re-examined in the context of computational networks, and a general learning method, computational network backpropagation, or CNBP, is derived.

Expert networks, or ENs, are the focus of the remainder of the chapter. An expert network is a symbolic-level computational network that can be derived from an expert system (ES). The network topology of the EN is derived from the rule base of the ES, the local processing functionality of the EN components from the rules of inference of the ES, and the global computation concepts of the EN from the inference engine of the ES. The process of constructing an EN from an ES is called *translation*. Translation, before or after learning, is a reversible process.

Learning methods for CNs can be instantiated for expert networks. In particular, CNBP specializes to expert network backpropagation, or ENBP, a learning method that has proven to be useful in knowledge acquisition and refinement. ENBP allows an expert system builder to make the transition from coarse knowledge, in the form of rough-draft rules, to fine knowledge, in the form of rules with subtlety represented by analog parameters such as certainty factors, using supervised learning and the historical record of expert behavior as a training set. This relieves the human expert whose knowledge is being captured from specifying any parameters such as probabilities or certainties.

We conclude with an investigation of how an EN can be given the structure of an artificial neural network. By a neural network, or NN, we mean a computational network consisting entirely of sub-symbolic processors such as linear/sigmoidal units. The nodes of an EN can, in principle, be represented by small NNs, and we investigate the practicality of this theory. We show in practice how such components can be constructed and determine optimal neural architectures for such components. In this way an expert network is given a realization as a neural network with a hierarchical topological organization: a sparsely interconnected ($O(n)$) collection of densely intraconnected ($O(n^2)$) neural nets.

This hierarchical sparse/dense EN/NN organization is analogous to biological neural organization. It captures two levels of knowledge: domain knowledge in the sparse superstructure and metaknowledge in the dense substructures. The sparse/dense architecture also scales much more comfortably than the dense $O(n^2)$ connectivity of, for example, feedforward NNs. Memory stability is supported by constructive EN learning methods. Using conservative estimates of 10^{10} neurons and 10^{13} synapses in the human cerebral cortex, and assuming a sparse/dense topology with constant size dense subnetworks, an estimated subnetwork size is $1,000$ units. This is more than enough resource to train complex symbolic-level components.

Research continues in this area. Projects using expert networks as a tool in large expert system development are testing the limits of usefulness of EN technology. Other more fundamental work investigates how dual sparse/dense representations of expert networks may self-organize from random soup of neural networks and may shed light on questions of the role of early learning in cognitive development.

Computational networks are ubiquitous in the natural world and in the creations of humankind.

REFERENCES

[1] B. G. Buchanan and E. H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.*

Addison-Wesley, Reading, MA., 1984.

[2] P. M. Churchland and P. S. Churchland. Could a machine think? *Scientific American*, 262:32-39, 1990.

[3] C. H. Dagli and R. Stacey. A prototype expert system for selecting control charts. *Int. J. Prod. Res.*, 26:987-996, 1988.

[4] S. P. Eberhardt, T. Daud, D. A. Kerns, T. X. Brown, and A. P. Thakoor. Competitive neural architecture for hardware solution to the assignment problem. *Neural Networks*, 4:431-442, 1991.

[5] S. P. Eberhardt, T. Duong, and A. P. Thakoor. Design of parallel hardware neural network systems from custom analog VLSI building block chips. In *Proceedings IJCNN 89 -Washington, DC*, volume 2, pages 183-190, Piscataway, NJ, 1989. IEEE.

[6] S. P. Eberhardt, T. Duong, and A. P. Thakoor. A VLSI building block chip for hardware neural network implementations. In *Proceedings Third Annual Parallel Processing Symposium*, volume 1, pages 257-267, Fullerton, CA, 1989. IEEE Orange County Computer Society.

[7] R. C. Eberhart and R. W. Dobbins. *Neural Network PC Tools*. Academic Press, San Diego, 1990.

[8] S. E. Fahlman and C. Lebiere. The cascade correlation learning architecture. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2, pages 524-532. Morgan Kaufmann, New York, 1990.

[9] W.-Z. Fang, S. I. Hruska, and R. C. Lacher. Expert networks: An empirical study of expert network backpropagation learning. 1994. in preparation.

[10] W. J. Freeman. Simulation of chaotic EEG patterns with a dynamic model of the olfactory system. *Biological Cybernetics*, 56:139-150, 1987.

[11] L.-M. Fu and L.-C. Fu. Mapping rule-based systems into neural architecture. In *Knowledge Based Systems*, volume 3, pages 48-56. 1990.

[12] K.-I. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183-192, 1989.

[13] S. I. Gallant. Connectionist expert systems. *Communications of the Association for Computing Machinery*, 24:152-169, 1988.

[14] J. Giarratano and G. Riley. *Expert Systems: Principles and Practice*. PWS-KENT, Boston, 1989.

[15] J. Giarratano and G. Riley. *Expert Systems: Principles and Practice.* PWS-KENT, Boston, 1994. Second Edition.

[16] L. O. Hall and S. G. Romaniuk. Fuzznet: Toward a fuzzy connectionist expert system development tool. In *Proceedings IJCNN 90 - Washington, DC)*, volume 2, pages 483–486, 1990.

[17] P. Harmon. G2: Gensym's real-time expert system. *Intelligent Software Strategies*, 9:1–14, 1993.

[18] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison-Wesley, New York, 1991.

[19] M. W. Hirsch. Convergent activation dynamics in continuous time networks. *Neural Networks*, 2:331–349, 1989.

[20] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings National Academy of Sciences, USA*, 79:1554-2558, 1982.

[21] J. J. Hopfield. Neurons with graded responses have collective computational properties like those of two-state neurons. *Proceedings National Academy of Sciences, USA*, 81:3088–3092, 1984.

[22] K. Hornik, M. Stinchcomb, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[23] S. I. Hruska and D. C. Kuncicky. Application of two-stage learning to an expert network for control chart selection. In C. Dagli, S. Kumara, and Y. Shin, editors, *Intelligent Engineering Systems Through Artificial Neural Networks*, pages 915–920. ASME Press, New York, 1991.

[24] S. I. Hruska and D. C. Kuncicky. Automated knowledge refinement for control chart selection. *Heuristics*, 1992.

[25] S. I. Hruska, D. C. Kuncicky, and R. C. Lacher. Hybrid learning in expert networks. In *Proceedings IJCNN 91 - Seattle*, volume 2, pages 117–120. IEEE 91CH3049-4, July 1991.

[26] S. I. Hruska, D. C. Kuncicky, and R. C. Lacher. Resuscitation of certainty factors in expert networks. In *Proceedings IJCNN 91 - Singapore*, pages 1653–1657. IEEE 91CH3065-0, November 1991.

[27] B. Kosko. *Neural Networks and Fuzzy Systems.* Prentice Hall, Englewood Cliffs, NJ, 1992.

[28] D. C. Kuncicky. The transmission of knowledge between neural networks and expert systems. In *WNN-AIND 91 (Proceedings of the First Workshop on Neural Networks)*, pages 311-319. Auburn University, 1990.

[29] D. C. Kuncicky. *Isomorphism of Reasoning Systems with Applications to Autonomous Knowledge Acquisition.* PhD thesis, Florida State University, Tallahassee, FL., 1991. R. C. Lacher, Major Professor.

[30] D. C. Kuncicky, S. I. Hruska, and R. C. Lacher. Shaping the behavior of neural networks. In *WNN-AIND 91 (Proceedings of the Second Workshop on Neural Networks)*, pages 173-180. Auburn University, SPIE Volume 1515, 1991.

[31] D. C. Kuncicky, S. I. Hruska, and R. C. Lacher. Hybrid systems: The equivalence of expert system and neural network inference. *International Journal of Expert Systems*, 4:281-297, 1992.

[32] R. C. Lacher. Node error assignment in expert networks. In A. Kandel and G. Langholz, editors, *Hybrid Architectures for Intelligent Systems*, pages 29-48. CRC Press, London, 1992.

[33] R. C. Lacher. The symbolic/sub-symbolic interface: Hierarchical network organizations for reasoning. In R. Sun, editor, *Integrating Neural and Symbolic Processes.* AAAI-92 Workshop, 1992.

[34] R. C. Lacher. Expert networks: Paradigmatic conflict, technological rapprochement. *Minds and Machines*, 3:53-71, 1993.

[35] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky. Expert networks: A neural network connection to symbolic reasoning systems. In M. B. Fishman, editor, *Proceedings FLAIRS 91*, pages 12-16, St. Petersburg, FL, 1991. Florida AI Research Society.

[36] R. C. Lacher, S. I. Hruska, and D. C. Kuncicky. Backpropagation learning in expert networks. *IEEE Transactions on Neural Networks*, 3:62-72, 1992.

[37] J. J. Mahoney and R. J. Mooney. Combining connectionist and symbolic learning to refine certainty-factor rule-bases. *Connection Science*, 5:339-364, 1993.

[38] J. J. Mahoney and R. J. Mooney. Modifying network architectures for certainty-factor rule-base revision. In *Proceedings International Symposium on Integrating Knowledge and Neural Heuristics 1994*, pages 75-84, Gainesville, FL 32609-3476, 1994. University of Florida DOCE.

[39] K. Narita and R. C. Lacher. The FEN learning architecture. In *Proceedings IJCNN 93 - Nagoya*, pages 1901-1905, Washington, DC, 1993. Institute of Electrical and Electronic Engineers.

[40] K. D. Nguyen, K. S. Gibbs, R. C. Lacher, and S. I. Hruska. A connection machine based knowledge refinement tool. In M. B. Fishman, editor, *FLAIRS 92*, pages 283-286, St. Petersburg, 1992. Florida Artificial Intelligence Research Symposium.

[41] T. Oi. Chaos dynamics executes inductive inference. *Biological Cybernetics*, 57:47-56, 1987.

[42] R. Ratliff. Continuous vs discrete information processing: Modelling the accumulation of partial information. *Psychological Review*, 95:238-255, 1988.

[43] R. R. Rocker. An event-driven approach to artificial neural networks. Master's thesis, Florida State University, Tallahassee, FL., 1991. S. I. Hruska, Major Professor.

[44] D. E. Rumelhart and J. L. McClelland. *Parallel Distributed Processing*. MIT Press, Cambridge, MA., 1986.

[45] M. J. Salzgeber, J. L. Franke, and S. I. Hruska. Managing uncertainty in clips: A system level approach. In M. B. Fishman, editor, *FLAIRS 93*, pages 142-146, St. Petersburg, 1993. Florida Artificial Intelligence Research Symposium.

[46] J. R. Searle. Is the brain's mind a computer program? *Scientific American*, 262:26-31, 1990.

[47] T. J. Sejnowski and C. R. Rosenberg. Parallel networks that learn to pronounce english text. *Complex Systems*, 1:145-168, 1987.

[48] E. H. Shortliffe. *Computer-Based Medical Consultations: MYCIN*. Elsevier, New York, 1976.

[49] E. H. Shortliffe and B. G. Buchanan. A model of inexact reasoning in medicine. In *Rule-Based Expert Systems*, pages 233-262. Addison-Wesley, New York, 1985.

[50] P. K. Simpson. *Artificial Neural Systems*. Pergamon Press, New York, 1990.

[51] C. Skarda and W. J. Freeman. How brains make chaos in order to make sense of the world. *Behavioral and Brain Sciences*, 10:161-195, 1987.

[52] R. Sun. A discrete neural network model for conceptual representation and reasoning. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Hillsdale, N.J., 1989. Erlbaum.

[53] R. Sun. On variable binding in connectionist networks. *Connection Science*, 4:93–124, 1992.

[54] V. S. Sunderam. Heterogeneous environments for network concurrent computing. *Journal of Future Generation Computer Systems*, 8:191–203, 1992.

[55] V. S. Sunderam and G. A. Geist. Network based concurrent computing on the pvm system. *Journal of Concurrency: Practice and Experience*, 4:293-311, 1992.

[56] G. G. Towell, J. W. Shavlik, and M. O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings AAAI-90*, pages 861–866, New York, 1990. Morgan Kaufmann.

[57] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

PART II

DISTRIBUTED ARCHITECTURES

Part II: Distributed Architectures

- Chapter 5 (by Risto Miikkulainen) presents a distributed connectionist model for parsing recursive relative clauses.
- Chapter 6 (by David Noelle and Gary Cottrell) presents a distributed connectionist model that learns to act in accordance with a given set of instructions.
- Chapter 7 (by Noel Sharkey and Stuart Jackson) challenge the notion that the precise distances between distributed representations in the hidden layer of a backpropagation network reflect systematic semantic and/or structural similarity relations.

5

Subsymbolic Parsing of Embedded Structures

RISTO MIIKKULAINEN

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712

1 INTRODUCTION

Symbolic artificial intelligence is motivated by the hypothesis that symbol manipulation is both necessary and sufficient for intelligence [34]. Symbolic systems have been quite successful, for example, in modeling in-depth natural language processing [13, 26, 43], episodic memory [22, 24], and problem solving [23, 35, 36]. In such systems, knowledge is encoded in terms of explicit symbolic structures, and processing is based on handcrafted rules that operate on these structures.

For cognitive processes based on conscious rule application, symbol manipulation is a natural approach. However, the symbolic approach does not naturally lend itself to modeling the statistical (intuitive) nature of certain cognitive processes. In many routine tasks, such as image understanding or sentence processing, large amounts of information about the current context and past experience are simultaneously brought together to produce the most appropriate interpretation. The process occurs through associations immediately, in parallel, and without conscious control.

Subsymbolic (i.e. distributed) neural networks represent knowledge in terms of correlations, coded in the weights of the network. For a given input situation, the network computes the most likely answer given its past training. The process is immediate and opaque, and fits very well into modeling intuitive inference (see also [18, 54, 61]). A major motivation for subsymbolic artificial intelligence, therefore, is to give a better account for high-level cognitive phenomena that are intuitive in nature. For example, the subsymbolic approach to sentence processing has several appealing properties: it is possible to combine syntactic, semantic, and thematic constraints in the interpretation, generate expectations automatically, generalize to new inputs, and process noisy sentences robustly [14, 15, 28, 30, 58]. To a limited extent, it is even possible to train such networks to process sentences with complex grammatical structure, such as embedded relative clauses [4, 21, 29, 50, 59].

However, it has been very difficult to build subsymbolic systems that could effectively deal with structure. Distributed neural networks are pattern transformers, and they generalize by interpolating between patterns on which they were trained. They cannot make inferences by dynamically combining knowledge about structures that were previously associated to different contexts. For example, a sentence processing network can be trained to form a case-role representation of each clause in a sentence like `The girl, who liked the dog, saw the boy`[1], and it will be able to generalize to different versions of the same structure, such as `The dog, who bit the girl, chased the cat` [29]. However, such network cannot parse sentences with novel combinations of familiar clause structures, such as `The girl, who liked the dog, saw the boy, who chased the cat`. Such a lack of generalization is a serious problem, given how effortlessly people can understand sentences they may have never seen before.

This chapter describes SPEC (Subsymbolic Parser for Embedded Clauses), a subsymbolic sentence parsing architecture that can generalize to new relative clause structures. The basic idea is to separate the tasks of segmenting the input word sequence into clauses, forming the case-role representations, and keeping track of the recursive embeddings into different modules. The system is trained with only the most basic relative clause constructs. It abstracts the "idea" of a relative clause from these examples and is able to generalize to novel sentences with remarkably complex structure. Importantly, although SPEC has powerful symbolic capabilities, it is not a neural network reimplementation of a symbol processor. It is purely a distributed system, and has the usual properties of such systems. For example, unlike symbolic parsers, the network exhibits plausible memory degradation as the depth of the center embeddings increases, and its performance is aided by semantic constraints between the constituents. Also, unlike many modular neural networks architectures, SPEC is self-contained. During performance, SPEC controls its own execution, and no external symbolic supervisor is needed.

[1] In all examples in this chapter, commas are used to indicate clause boundaries for clarity.

2 Overview of Subsymbolic Sentence Processing

Sentence processing has been an active area of connectionist research for about a decade. Subsymbolic models have been developed to address a variety of issues such as semantic interpretation, learning syntax and semantics, prepositional phrase attachment, anaphora resolution, active-passive transformation, and translation [1, 2, 7, 8, 10, 25, 33, 61].

A good amount of work has been done showing that networks can capture grammatical structure. For example, Servan-Schreiber et al. [46, 47] showed how Simple Recurrent Networks (SRNs; [14]) can learn a finite state grammar. In an SRN, the pattern in the hidden layer is copied to the previous-hidden-layer assembly and serves as input to the hidden layer during the next step in the sequence, thus implementing a sequence memory. The network is trained with examples of input/output sequences, adjusting all forward weights according to the backpropagation algorithm [42]. Servan-Schreiber et al. trained an SRN with sample strings from a particular grammar, and it learned to indicate the possible next elements in the sequence. For example, given a sequence of distributed representations for elements B, T, X, X, V, and V, the network turns on two units representing X and S at its localist output layer, indicating that in this grammar, the string can continue with either X or S.

Elman [15, 16] used the same network architecture to predict a context-free language with embedded clauses. The network could not learn the language completely, but its performance was remarkably similar to human performance. It learned better when it was trained incrementally, first with simple sentences and gradually including more and more complex examples. The network could maintain contingencies over embeddings if the number of intervening elements was small. However, deep center embeddings were difficult for the network, as they are for humans.

The above architectures demonstrated that distributed networks build meaningful internal representations when exposed to examples of strings in a language. They did not address how such capabilities could be put to use in parsing and understanding language. McClelland and Kawamoto [28] identified the sentence case-role assigment as a good approach. Case-role representation is a common artificial intelligence technique for describing the shallow semantic meaning of a sentence. The idea is loosely based on the theory of thematic case roles [17, 9]. Each act is described by the main verb and a set of semantic cases such as agent, patient, instrument, location, and recipient. The task is to decide which constituents fill these roles in the sentence. The approach is particularly well-suited for neural networks because the cases can be conveniently represented as assemblies of units that hold distributed representations, and the parsing task becomes that of mapping between distributed representation patterns. McClelland and Kawamoto showed that given the syntactic role assignment of the sentence as the input, the network could assign the correct case roles for each constituent. The network also automatically performed semantic enrichment on the word representations (which were hand-coded concatenations of binary semantic features), and disambiguated between the different senses of ambiguous words.

Miikkulainen and Dyer [31, 32] showed that essentially the same task can be performed from sequential word-by-word input by a simple recurrent network, and, through a technique called FGREP (Forming Global Representations with Extended backPropagation), meaningful distributed representations for the words can be automatically developed at the same time. In FGREP, the component values are assigned initially randomly within $[0, 1]$ and modified by backpropagation as part of learning the task. The final representations reflect how the words are used in the examples, and in that sense, represent word meanings. Systems with FGREP representations generally have a strong representation of context, which results in good generalization properties, robustness against noise and damage, and automatic "filling in" of missing information.

St. John and McClelland [57, 58] further explored the subsymbolic approach to sentence interpretation in their Sentence Gestalt model. They aimed at explaining how syntactic, semantic, and thematic constraints are combined in sentence comprehension, and how this knowledge can be coded into the network by training it with queries. The gestalt is a hidden-layer representation of the whole sentence, built gradually from a sequence of input words by a simple recurrent network. The second part of the system (a three-layer backpropagation network) is trained to answer questions about the sentence gestalt, and in the process, useful thematic knowledge can be injected into the system.

The above three parsing architectures each built a semantic interpretation of the sentence, but they could not handle grammatically very complex sentences. Several extensions and some completely new architectures that could do that have been proposed. For example, the CLAUSES system [29] was an extension of the SRN+FGREP case-role assignment architecture into sentences with multiple clauses. CLAUSES read clause fragments one at a time, brought together the separated constituents, and concatenated the case-role representations into a comprehensive canonical sentence representation in its output layer. CLAUSES was limited both by the rigid output representation and also by a somewhat surprising lack of generalization into new sentence structures. On the other hand, Stolcke [59] showed that if the output representation was made more flexible, the network was likely to forget earlier constituents. The conclusion from these two models is that straightforward applications of simple recurrent networks are unlikely to be successful in parsing and representing grammatical structure.

A number of researchers have proposed modular and more structured architectures. In Jain's [21] Structured Incremental Parser, one module was trained to assign words into phrases, and another to assign phrases into case roles. These modules were then replicated multiple times so that the recognition of each constituent was guaranteed independent of its position in the sentence. In the final system, words were input one at a time, and the output consisted of local representations for the possible assignments of words into phrases, phrases into clauses, phrases into roles in each clause, and for the possible relationships of the clauses. A consistent activation of the output units represented the interpretation of the sentence. The system could interpret complicated sentence structures, and even ungrammatical and incomplete input. However, it did not build an explicit representation for the sentence meaning. The parse result was a description of the semantic relations of the constituents; the constituents themselves were not represented.

Berg's XERIC [4] and Sharkey and Sharkey's parser [50] were both based on the idea of combining a simple recurrent network with a Recursive Auto-Associative Memory (RAAM; [39]) that encodes and decodes parse trees. RAAM is a three-layer backpropagation network trained to perform an identity mapping from input to output. As a side effect, the hidden layer learns to form compressed representations of the network's input/output patterns. These representations can then be recursively used as constituents in other input patterns. A potentially infinite hierarchical data structure, such as a parse tree, can this way be compressed into a fixed-size representation. The structure can later be reconstructed by loading the compressed representations into the hidden layer and reading off the expanded representation at the output.

In Sharkey and Sharkey's model, first the RAAM network was trained to form compressed representations of syntactic parse trees. Second, an SRN network was trained to predict the next word in the sequence of words that make up the sentence. Third, a standard three-layer feedforward network was trained to map the SRN hidden-layer patterns into the RAAM parse-tree representations. During performance, a sequence of words was first read into the SRN, its final hidden layer transformed into a RAAM hidden layer, and then decoded into a parse tree with the RAAM network. Berg's XERIC worked in a similar manner, except the SRN hidden layer representations were directly decoded by the RAAM network.

All five of the above architectures can parse sentences with complex grammatical structure, and they can generalize to new sentences where constituents have been substituted with other familiar constituents. Unfortunately, generalization into new sentence structures is limited. For example, due to its rigid output representation and excessive context-sensitivity, CLAUSES could not parse `The girl, who liked the dog, saw the boy, who chased the cat`, even if it knew how to process `The girl, who liked the dog, saw the boy` and `The girl saw the boy, who chased the cat`. Jain's architecture is similarly limited because of the fixed hardware constraints; XERIC and Sharkey and Sharkey's parser because the RAAM architecture generalizes poorly to new tree structures.

The architecture described in this chapter, SPEC, was especially designed to address the problem of generalization into new sentence structures. SPEC is a descendant of CLAUSES. The central component is the familiar simple recurrent network that reads distributed word representations as its input and generates case-role representations as its output. SPEC's generalization capability is based on simplifying the SRN's task through three architectural innovations: (1) training the SRN to generate a sequence of clause case-role representations as its output (like [59]) instead of a single comprehensive representation, (2) introducing a segmenter network that breaks the input sequence into smaller chunks, and (3) introducing a stack network that memorizes constituents over intervening embedded clauses. Below, the SPEC architecture is described in detail, and its performance is demonstrated on an artificially-generated corpus of sentences with complex relative clause structures.

3 THE SPEC ARCHITECTURE

An overview of the architecture is shown in figure 1. The system receives a sequence of word representations as its input, and for each clause in the sentence, forms an output representation indicating the assignment of words into case roles. The case-role representations are read off the system and placed in a short-term memory (currently outside SPEC) as soon as they are complete.

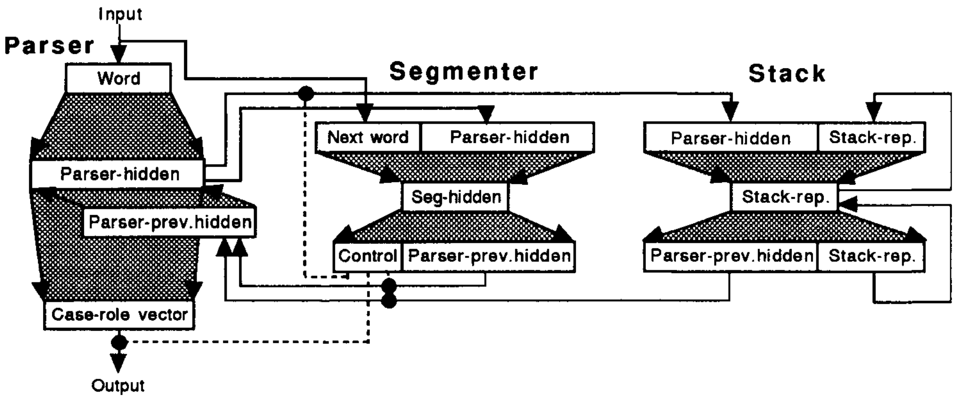


Figure 1 The SPEC sentence processing architecture. The system consists of the Parser (a simple recurrent network), the Stack (a RAAM network), and the Segmenter (a feedforward network). The gray areas indicate propagation through weights, the solid lines stand for pattern transport, and the dashed lined represent control outputs (with gates). The lines controlling propagation within the Stack have been omitted.

The collection of case-role representations constitutes the final result of the parse. This is a canonical representation for the sentence. The recursive clause structure is not explicitly represented, but it is implicit in the clause representations. For example, two clauses may share the same agent, or the agent of one clause may be the patient of another clause. The idea behind such representation is that the recursive structure is a property of the language, not the information itself. The canonical representation can serve as input to higher-level cognitive processes, which can access all constituents in parallel without being biased by the linguistic form of the information.

SPEC consists of three main components: the Parser, the Segmenter, and the Stack. Below, each component is described in detail and the reasons for the main architectural choices are explained.

3.1 THE PARSER

The Parser performs the actual transformation of the word sequence into the case-role representations, and like most of the other parsers described above, it is based on the simple recurrent network architecture (figure 2). Words are

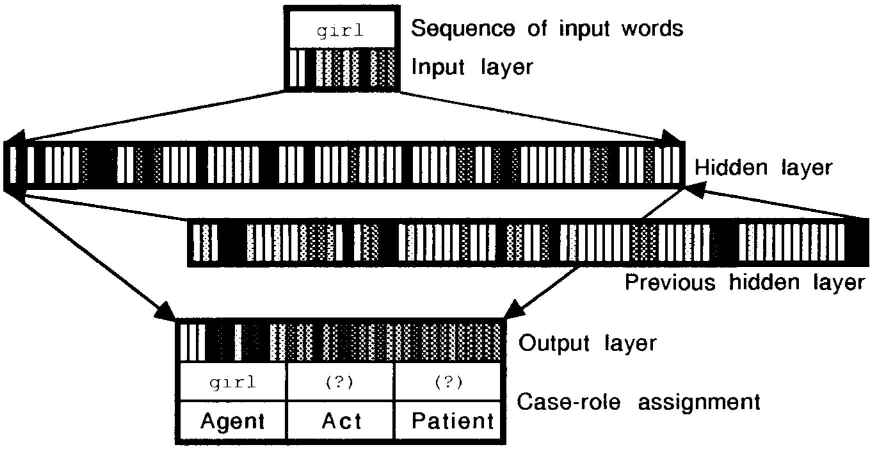


Figure 2 **The Parser network.** The figure depicts a snapshot of the network after it has read the first two words `The` and `girl`. The activity patterns in the input and output assemblies consist of word representations. The input layer holds the representation for the last word, `girl`, and the activity pattern at the output represents the (currently incomplete) case-role assignment of the clause. At this point, it is clear that `girl` is going to be the agent. The act and the patient are not known; the patterns in these slots indicate expectations, that is, averages of all possible alternatives.

represented distributively as vectors of gray-scale values between 0 and 1. The component values are initially assigned randomly and modified by the FGREP method [30, 31, 32] as part of the learning process. FGREP is a convenient way to form distributed representations for input/output items, but SPEC is not dependent on FGREP. The word representations could have been obtained through semantic feature encoding as well (as was done by e.g. McClelland and Kawamoto [28]). SPEC will even work with random word representations, although some of the advantages of distributed representations (such as generalization, robustness, and context representation) would not be as strong.

The case-role assignment is represented at the output of the Parser as a case-role vector (CRV), that is, a concatenation of those three word representation vectors that fill the roles of agent, act, and patient in the sentence[2] (figure 2).

[2]The representation was limited to three roles for conciseness; more roles could be easily included. However, each slot can represent only one constituent at any one time.

For example, the word sequence `the girl saw the boy` receives the case-role assignment agent=`girl`, act=`saw`, patient=`boy`, which is represented as the vector `|girl saw boy|` at the output of the Parser network. When the sentence consists of multiple clauses, the relative pronouns are replaced by their referents: `The girl, who liked the dog, saw the boy` parses into two CRVs: `|girl liked dog|` and `|girl saw boy|`.

The obvious approach for representing multiple CRVs would be to concatenate them into a single vector at the output of the Parser network. This was the approach taken in CLAUSES [29]. Such representation has two serious limitations:

1. The size of the output layer always poses a hard limit on the number of clauses in the sentence. If there is space for three CRVs, sentences with four clauses (such as `The girl saw the boy, who chased the cat, who saw the girl, who liked the dog`) could not be parsed without changing the architecture and retraining the entire network.
2. Somewhat less obviously, such representation turns out to be detrimental to generalization. The network always has to represent the entire sentence in its memory (in the hidden layer). Every new item in the sequence is interpreted in the context of the entire sequence so far. CLAUSES learned to recognize certain sequences of act fragments, and to associate a particular interpretation to each sequence. If there ever was a novel input, such as an additional tail embedding in the end of an otherwise familiar sequence, the network did not know how to combine it with its current hidden-layer representation. As a result, CLAUSES could only process variations of those clause structures it was trained on.

The above problems can be overcome if the network is not required to form a complete sentence representation at its output. Instead, the network generates the CRV for each clause as soon as the information for the clause is complete. Another network (or even a symbolic system [53]) then reads the sequence of complete act representations as its input and builds a representation for the whole sentence using a flexible-size representation technique, such as tensor-product encoding [12, 55].

This is the approach taken in SPEC. The Parser receives a continuous sequence of input word representations as its input, and its target pattern changes at each clause boundary. For example, in reading `The girl, who liked the dog, saw the boy`, the target pattern representing `|girl saw boy|`

is maintained during the first two words, then switched to |girl liked dog| during reading the embedded clause, and then back to |girl saw boy| for the rest of the sentence. The CRV for the embedded clause is read off the network after dog has been input, and the CRV for the main clause after the entire sentence has been read.

When trained this way, the network does not have to maintain information about the entire past input sequence in its memory, making it possible in principle to generalize to new clause structures. The early words do in fact fade from the memory as more words are read in, but by itself this effect is not strong enough, and needs to be enforced by an additional network (the Segmenter, discussed in section 3.3). However, even such slight forgetting is strong enough to cause problems with the center embeddings. After parsing who liked the dog, the network does not remember that it was the girl who saw the boy. The system needs a memory component external to the parser so that the top-level parse state can be restored before reading rest of the top-level constituents. This is the task of the Stack network.

3.2 THE STACK

The hidden layer of a simple recurrent network forms a compressed description of the sequence so far. The Stack has the task of storing this representation at each center embedding, and restoring it upon return from the embedding. For example, in parsing The girl, who liked the dog, saw the boy, the hidden-layer representation is pushed onto the stack after The girl, and popped back to the Parser's previous-hidden-layer assembly after who liked the dog. In effect, the SRN can then parse the top-level clause as if the center embedding had not been there at all.

The Stack is implemented as a RAAM network [39] trained to encode and decode linear lists (figure 3). The input/output of the Stack consists of the Stack's top element and the compressed representation for the rest of the stack. Initially the stack is empty, which is represented by setting all units in the "Stack" assembly to 0.5 (figure 3). The first element, such as the hidden-layer pattern of the Parser network after reading The girl, is loaded into the "Push" assembly, and the activity is propagated to the hidden layer. The hidden-layer pattern is then loaded into the "Stack" assembly at the input, and the Stack network is ready for another push operation.

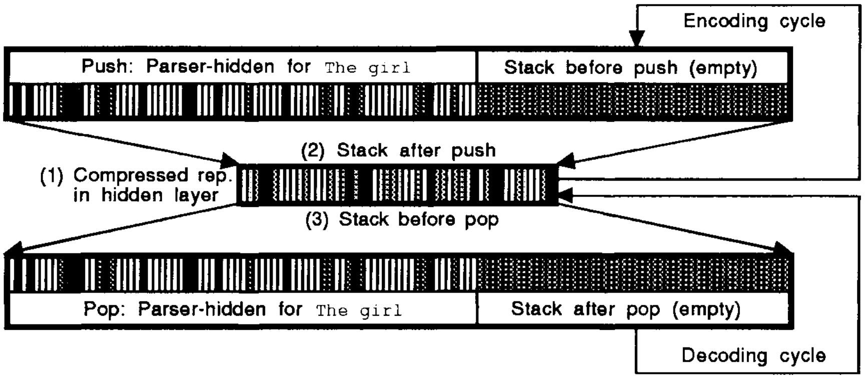

Encoding cycle
Push: Parser-hidden for The girl
Stack before push (empty)
(2) Stack after push
(1) Compressed rep. in hidden layer
(3) Stack before pop
Pop: Parser-hidden for The girl
Stack after pop (empty)
Decoding cycle

Figure 3 The Stack network. This figure simultaneously illustrates three situations that occur at different times during the training and the performance of the Stack: (1) A training situation where the network learns to autoassociate an input pattern with itself, forming a compressed representation at the hidden layer; (2) A push operation, where a representation in the "Push" assembly is combined with the empty-stack representation (in the "Stack" assembly) to form a compressed representation for the new stack in the hidden layer; (3) A pop operation, where the current stack representation in the hidden layer generates an output pattern with the top element of the stack in the "Pop" assembly and the representation for the remaining stack (currently empty) in the "Stack" assembly.

When the Parser returns from the center embedding, the stored pattern needs to be popped from the stack. The current stack representation is loaded into the hidden layer, and the activity is propagated to the output layer. At the output, the "Pop" assembly contains the stored Parser-hidden-layer pattern, which is then loaded into the previous-hidden-layer assembly of the Parser network (figure 1). The "Stack" assembly contains the compressed representation for the rest of the stack, and it is loaded to the hidden layer of the Stack network, which is then ready for another pop operation.

RAAM networks usually generalize well into encoding and decoding new instances of familiar structures, but poorly into processing new structures [5, 7, 8, 50]. The deeper the structure, the less accurate its representation, because more and more information will be superimposed on the same fixed-width vector. Fortunately, this is not a major problem for SPEC, because the RAAM network only needs to encode one type of structure (a linear list), and there are very strong memory limitations in human processing of deep embedded structures as well. It should be very easy to train the RAAM network to model human memory for embedded clauses, and it should generalize well to new instances.

3.3 THE SEGMENTER

The Parser+Stack architecture alone is not quite sufficient for generalization into novel relative clause structures. For example, when trained with only examples of center embeddings (such as the above) and tail embeddings (like `The girl saw the boy, who chased the cat`), the architecture generalizes well to new sentences such as `The girl, who liked the dog, saw the boy, who chased the cat.` However, the system still fails to generalize to sentences like `The girl saw the boy, who the dog, who chased the cat, bit.` The problem is the same as with CLAUSES: even though the Stack takes care of restoring the earlier state of the parse, the Parser has to learn all the different transitions into the relative clauses. If it has encountered center embeddings only at the beginning of the sentence, it cannot generalize to a center embedding that occurs after an entire full clause has already been read. Even though the Parser is free to "forget" the irrelevant information in the early sequence, the hidden-layer patterns remain sufficiently different so that its processing knowledge does not carry over.

The solution is to train an additional network, the Segmenter, to divide the input sequence into clauses. The segmenter receives the current hidden-layer pattern as its input, together with the representation for the next input word, and it is trained to produce a modified hidden-layer pattern as its output (figure 4). The output is then loaded into the previous-hidden-layer assembly of the Parser. In the middle of reading a clause, the Segmenter passes the hidden-layer pattern through without modification. However, if the next word is a relative pronoun, the segmenter modifies the pattern so that only the relevant information remains. In the above example, after `boy` has been read and `who` is next to come, the Segmenter generates a pattern similar to that of the Parser's hidden layer after only `The boy` in the beginning of the sentence has been input.

In other words, the Segmenter (1) detects transitions to relative clauses, and (2) changes the sequence memory so that the Parser only has to deal with one type of clause boundary. This way, the Parser's task becomes sufficiently simple so that the entire system can generalize to new structures. The Segmenter plays

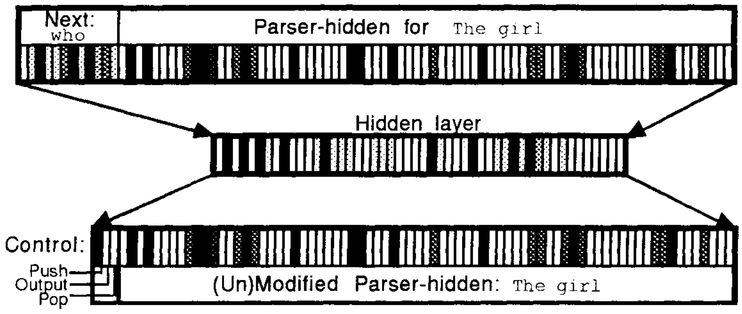

Next:
who
Parser-hidden for The girl
Hidden layer
Control:
Push
Output
Pop
(Un)Modified Parser-hidden: The girl

Figure 4 The Segmenter network. The Segmenter receives the Parser's hidden-layer pattern as its input together with the next input word, which in this case is `who`. The control outputs are 1, 0, 0, indicating that the Parser's hidden-layer representation should be pushed onto the Stack, the current case-role representation is incomplete and should not be passed on to the output of the system, and the stack should not be popped at this point. In this case, the Segmenter output is identical to its input, because `the girl` is the smallest context that the Parser needs to know when entering a center embedding.

a central role in the architecture. The next section shows that it is very natural to give the Segmenter a complete control over the entire parsing process.

3.4 CONTROL

At first glance, the control of execution in SPEC seems rather complicated. The activation patterns propagate between networks in a very specific manner, and execution of each network needs to be carefully timed with respect to what the other networks are doing. However, it is actually very easy to train the Segmenter to control the parsing process. The Segmenter always sees the current state of the parse (as encoded in the hidden layer of the Parser network) and the incoming word, and based on this information, it can control the pathways of the system. There are five different control tasks in the SPEC system:

1. Detecting clause transitions and modifying the sequence memory to remove unnecessary previous context as described above.

2. Recognizing the end of the sentence, indicated by "." (full stop) in the input sequence, and subsequently clearing the previous hidden layer (which is all-0 at the beginning of each sentence). This makes it possible for the system to parse multiple sentences without an external "reset".
3. Deciding when to push the Parser's hidden-layer representation onto the stack. This requires opening the pathway from the hidden layer to the "Push" assembly of the Stack, allowing propagation to the Stack's hidden layer, and transporting the resulting pattern back to the Stack's input assembly.
4. Deciding when to pop the previous hidden layer from the stack; this task involves allowing propagation from the Stack's hidden layer to its output layer, transporting the output "Stack" pattern back to its hidden layer, and opening the pathway from the "Pop" assembly to the Parser's previous hidden layer.
5. Deciding when the Parser's output CRV is complete, and consequently, opening the output pathway to the external short-term memory system.

Control is implemented through three additional units at the Segmenter's output (figure 4). These are called Push, Pop, and Output, corresponding to the tasks 3, 4, and 5 above. These units gate the system pathways through multiplicative connections (as described in [38, 41]). The weights on the pathways are multiplied by the output values, so that propagation only takes place when the output is high. The Segmenter is trained to output 1 for the desired propagation, and 0 otherwise.

The control implementation in SPEC emphasizes an important point: although much of the structure in the parsing task is programmed into the system architecture, SPEC is still a self-contained distributed neural network. In many modular neural network architectures control is due to a hidden symbolic supervisor. SPEC demonstrates that such external control mechanisms are not necessary: even a rather complex subsymbolic architecture can take care of its own control and operate independently of its environment.

4 EXPERIMENTS

A prototype implementation of SPEC was tested with an artificially-generated corpus of relative clause sentences. The purpose was to evaluate the soundness

```
S     →  NP VP "."
NP    →  DET N | DET N RC
VP    →  V NP
RC    →  who VP | who NP V
N     →  boy | girl | dog | cat
V     →  chased | liked | saw | bit
DET   →  the
```

Table 1 The sentence grammar.

of the basic ideas, test the cognitive plausibility of the model, and get a feeling for the scale-up possibilities of the approach. The experiments are described below, and some general conclusions drawn from them are presented in the Discussion section.

4.1 DATA

The training and testing corpus was generated from a simple phrase structure grammar depicted in table 1. This grammar generates sentences where each clause consists of three constituents: the agent, the verb and the patient. A relative `who`-clause could be attached to the agent or to the patient of the parent clause, and `who` could fill the role of either the agent or the patient in the relative clause. In addition to `who`, `the` and "." (full stop, the end-of-sentence marker that had its own distributed representation in the system just like a word), the vocabulary consisted of the verbs `chased`, `liked`, `saw` and `bit`, and the nouns `boy`, `girl`, `dog` and `cat`.

A number of semantic restrictions were imposed on the sentences. A verb could have only certain nouns as its agent and patient (see table 2). These restrictions are not necessary to train SPEC, but they create enough differences in the word usage so that their FGREP representations do not become identical [32, 30]. The main motivation for the restrictions, however, was to determine whether SPEC would be able to use the semantics to aid parsing under difficult conditions. The grammar was used to generate all sentences with up to four clauses, and those that did not match the semantic restrictions were discarded. The final corpus consists of 49 different sentence structures, with a total of 98,100 different sentences (table 3).

Verb	Case-role	Possible fillers
chased	Agent:	boy,girl,dog,cat
	Patient:	cat
liked	Agent:	boy,girl
	Patient:	boy,girl,dog
saw	Agent:	boy,girl,cat
	Patient:	boy,girl
bit	Agent:	dog
	Patient:	boy,girl,dog,cat

Table 2 Semantic restrictions.

Since the SPEC architecture divides the sentence parsing task into low-level pattern transformation, segmentation, and memory, each component needs to see only its own basic constructs during training. The combined architecture then forces generalization into novel combinations of these structures. The Parser and the Segmenter need to be able to process the following three types of sequences:

(1) `The girl saw the boy...` (top level clause)
(2) `...the girl, who saw the boy,...` (`who` as the agent)
(3) `...the girl, who the boy saw,...` (`who` as the patient).

The Segmenter also needs to see four different types of embedded clause transitions, such as

(1) `The girl, who...` (top-level center)
(2) `...the girl, who the boy, who...` (embedded center)
(3) `The girl saw the boy, who...` (top-level tail)
(4) `...the girl, who saw the boy, who...` (embedded tail),

and examples of the two different types of popping operations:

(1) `...the girl, who saw the boy, liked...` (after `who` as agent)

(2) `...the girl, who the boy saw, liked...` (after `who` as patient).

The Stack needs to handle only a very small number of different types of patterns for pushing and popping. Either it receives a center embedding at the top level, followed by a number of center embeddings at deeper levels, such as

```
(1) The girl,      (top-level center embedding)
    who the dog,   (first deeper center embedding)
    who the boy,   (second deeper center embedding)
    ...,
```

or it receives a number of deeper center embeddings without a preceding top-level embedding:

```
(2)  The girl saw the boy, who the cat,  (first deeper embedding)
     who the dog,                        (second deeper embedding)
     ...
```

Because the Segmenter makes all the clause transitions look the same for the Parser, the representations that are pushed on the stack are similar at all levels of embeddings. Therefore, if the Stack is trained to encode, say, a stack of 15 elements, it should generalize to the 16th push without any problems. However, three levels of center embeddings is about the most that would occur in a natural language, and as a result, the architecture cannot really make use of the generalization capabilities of the Stack. The Stack will not generalize to encoding and decoding a 3-element stack after it has been trained only up to 2-element stacks, and there is little point in doing that anyway. It is quite easy to train the Stack to up to 3 levels of embeddings and thereby guarantee that the Stack is not going to be limiting the generalization capabilities of the system.

4.2 Training Methodology

There is a variety of strategies for training a modular system such as SPEC. They usually lead to comparable results, but vary in amount of computational and programming effort involved, final accuracy, and robustness of the trained system.

Template	Generates	Example sentence
1.	20	The girl saw the boy.
2.	102	The girl saw the boy, who chased the cat.
3.	528	The girl saw the boy, who chased the cat, who saw the girl.
4.	2738	The girl saw the boy, who chased the cat, who saw the girl, who liked the dog.
5.	2814	The girl saw the boy, who chased the cat, who saw the girl, who the dog bit.
*6.	544	The girl saw the boy, who chased the cat, who the dog bit.
7.	2878	The girl saw the boy, who chased the cat, who the dog, who bit the girl, bit.
8.	2802	The girl saw the boy, who chased the cat, who the dog, who girl liked, bit.
9.	106	The girl saw the boy, who the dog bit.
10.	560	The girl saw the boy, who the dog, who chased the cat, bit.
11.	2878	The girl saw the boy, who the dog, who chased the cat, who saw the girl, bit.
12.	2962	The girl saw the boy, who the dog, who chased the cat, who the girl chased, bit.
13.	544	The girl saw the boy, who the dog, who the girl liked, bit.
14.	2898	The girl saw the boy, who the dog, who the girl, who chased the cat, liked, bit.
15.	2814	The girl saw the boy, who the dog, who the girl, who the cat saw, liked, bit.
16.	106	The girl, who liked the dog, saw the boy.
17.	544	The girl, who liked the dog, saw the boy, who chased the cat.
18.	2814	The girl, who liked the dog, saw the boy, who chased the cat, who saw the girl.
19.	2898	The girl, who liked the dog, saw the boy, who chased the cat, who the dog bit.
20.	560	The girl, who liked the dog, saw the boy, who the dog bit.

(to be continued on the next page)

Table 3 The sentence structures. The total number of sentences for each different clause structure is given together with an example sentence. The different clause structures are referred to as "sentence templates" below. SPEC was trained with 100 sentences from templates 6 and 40 each (with complete training of the Stack to up to three levels) and it generalized correctly to all others. Commas are inserted in the examples to help discern the clause boundaries; they were not part of the actual input.

Template	Generates	Example sentence
21.	2962	The girl, who liked the dog, saw the boy, who the dog, who chased the cat, bit.
22.	2878	The girl, who liked the dog, saw the boy, who the dog, who the boy liked, bit.
23.	544	The girl, who liked the dog, who bit the cat, saw the boy.
24.	2802	The girl, who liked the dog, who bit the cat, saw the boy, who chased the cat.
25.	2878	The girl, who liked the dog, who bit the cat, saw the boy, who the dog bit.
26.	2814	The girl, who liked the dog, who bit the cat, who saw the girl, saw the boy.
27.	2898	The girl, who liked the dog, who bit the cat, who the boy chased, saw the boy.
28.	560	The girl, who liked the dog, who the dog bit, saw the boy.
29.	2878	The girl, who liked the dog, who the dog bit, saw the boy, who chased the cat.
30.	2962	The girl, who liked the dog, who the dog bit, saw the boy, who the dog bit.
31.	2962	The girl, who liked the dog, who the dog, who chased the cat, bit, saw the boy.
32.	2878	The girl, who liked the dog, who the dog, who the boy liked, bit, saw the boy.
33.	102	The girl, who the dog bit, saw the boy.
34.	528	The girl, who the dog bit, saw the boy, who chased the cat.
35.	2738	The girl, who the dog bit, saw the boy, who chased the cat, who saw the girl.
36.	2814	The girl, who the dog bit, saw the boy, who chased the cat, who the dog bit.
37.	544	The girl, who the dog bit, saw the boy, who the dog bit.
38.	2878	The girl, who the dog bit, saw the boy, who the dog, who chased the cat, bit.
39.	2802	The girl, who the dog bit, saw the boy, who the dog, who the girl liked, bit.
*40.	544	The girl, who the dog, who chased the cat, bit, saw the boy.
41.	2814	The girl, who the dog, who chased the cat, bit, saw the boy, who liked the girl.
42.	2898	The girl, who the dog, who chased the cat, bit, saw the boy, who the girl liked.
43.	2802	The girl, who the dog, who chased the cat, who saw the boy, bit, saw the boy.
44.	2878	The girl, who the dog, who chased the cat, who the boy chased, bit, saw the boy.
45.	528	The girl, who the dog, who the boy liked, bit, saw the boy.
46.	2738	The girl, who the dog, who the boy liked, bit, saw the boy, who the dog bit.
47.	2814	The girl, who the dog, who the boy liked, bit, saw the boy, who chased the cat.
48.	2814	The girl, who the dog, who the boy, who chased the cat, liked, bit, saw the boy.
49.	2738	The girl, who the dog, who the boy, who the cat saw, liked, bit, saw the boy.
Total	98100	

Table 3 *(continued)* **The sentence structures.**

One possibility is to train the entire SPEC as a whole, propagating the patterns between modules as during normal performance. For example, the output of the Stack would be propagated into the previous-hidden-layer assembly of the Parser as it is, even if it is highly inaccurate during early training. The advantage is that the modules learn to compensate for each other's errors, and final accuracy may be better. On the other hand, convergence is often slower, because the modules have to continuously adjust to each other's changing output representations.

If SPEC is to be trained as a whole, a set of templates from table 3 must be selected so that all the basic constructs are included in the set of sentences. One such set consists of templates 3, 15, and 49. Indeed, trained with 100 randomly chosen examples from each template, the network correctly generalized to all other sentences in the entire corpus.

On the other hand, each component can be trained separately, with compatible training data from the same set of examples but without propagating the actual output to the input of the next network. For example, after the previous-hidden-layer representation is obtained from the stack, it is cleaned up (i.e. replaced by the correct representation) before actually loading it into the previous hidden layer. This way the modules learn more independently, and converge faster. If the Parser is trained first, the Segmenter and the Stack can be trained very efficiently with the Parser's final hidden-layer patterns. The total training time in CPU cycles is minimized this way. It is also possible to train the different networks simultaneously on separate machines, thereby minimizing the wallclock training time. In the end, after the networks have learned to produce output close to their targets, they can be connected and they will work well together, even filter out each other's noise [30].

Training SPEC is not computationally very intensive with this particular corpus, and therefore, the most convenient training strategy was selected for the experiments reported below. All modules were trained separately and simultaneously on a single machine, sharing the gradually evolving word and hidden-layer representations. With this strategy, it is enough to train SPEC only with templates 6 and 40, because they contain all the basic constructs for the Parser and the Segmenter. Complete training data for the Stack can be obtained from Parser's hidden layer during the course of processing sentences 6 and 40.

4.3 Results

The word representations consisted of 12 units. Parser's hidden layer was 75 units wide, that of the Segmenter 50 units, and that of the Stack 50 units. All networks were trained with plain on-line backpropagation with 0.1 learning rate and without momentum. The training set consisted of 100 randomly-selected sentences from templates 6 and 100 each. Both the Parser and the Segmenter developed word representations at their input layers (with a learning rate of 0.001). The Stack was trained to encode and decode up to three levels of center embeddings.

The convergence was very strong. After 400 epochs, the average error per output unit was 0.018 for the Parser, 0.008 for the Segmenter (0.002 for the control outputs), and 0.003 for the Stack, while an error level of 0.020 usually results in acceptable performance in similar assembly-based systems [30]. The training took approximately three hours on an IBM RS6000 workstation. The final representations, developed by FGREP, reflected the word categories very well.

SPEC's performance was then tested on the entire corpus of 98,100 sentences. The patterns in the Parser's output assemblies were labeled according to the nearest representation in the lexicon. The control output was taken to be correct if those control units that should have been active at 1 had an activation level greater than 0.7, and those that should have been 0 had activation less than 0.3. Measured this way, the performance was excellent: SPEC did not make a single mistake in the entire corpus, neither in the output words or in control. The average unit error was 0.034 for the Parser, 0.009 for the Segmenter (0.003 for control), and 0.005 for the Stack. There was very little variation between templates and words within each sentence, indicating that the system was operating within a safe margin.

The main result, therefore, is that the SPEC architecture successfully generalizes not only to new instances of the familiar sentence templates, but to new templates as well, which the earlier sentence processing architectures such as CLAUSES could not do. However, SPEC is not a mere reimplementation of a symbol processor. As SPEC's Stack becomes increasingly loaded, its output becomes less and less accurate; symbolic systems do not have any such inherent memory degradation. An important question is, does SPEC's performance degrade in a cognitively plausible manner, that is, does the system have similar difficulties in processing recursive structures as people do?

There are two ways to elicit enough errors from SPEC to analyze its limitations: (1) it can be tested during early training, or (2) its memory can be disturbed by noise. In a sense, testing during training illustrates developmental effects, whereas adding noise can be claimed to simulate overload, stress, cognitive impairment, and lack of concentration situations. Both methods produce similar results; ones obtained with noise are reported below.

The Stack's performance was degraded by adding 30% noise in its propagation. During encoding, the final value h_i of the hidden unit i was obtained from r_i, the value after correct propagation, by the transformation

$$h_i = 0.70r_i + 0.30X, \tag{5.1}$$

where X is a random variable uniformly distributed within [0, 1]. Similarly during decoding, the output values o_i were degraded by

$$o_i = 0.70c_i + 0.30X, \tag{5.2}$$

where c_i is the correct value of unit i. The SPEC system turned out to be remarkably robust against such degradation. The average Parser error rose to 0.058, but the system still got 94% of its output words right, with very few errors in control.

As expected, most of the errors occurred as a direct result of popping back from center embeddings with an inaccurate previous-hidden-layer representation. For example, in parsing `The girl, who the dog, who the boy, who chased the cat, liked, bit, saw the boy` (template 48), SPEC would have trouble remembering the agents of `liked`, `bit` and `saw`, and patients of `liked` and `bit`. The performance depends on the level of the embedding in an interesting manner. It is harder for the network to remember the earlier constituents of shallower clauses than those of deeper clauses (figure 5). For example, SPEC could usually connect `boy` with `liked`, but it was harder for it to remember that it was the `dog` who `bit` and the `girl` who `saw` in the above example.

Such behavior seems plausible in terms of human performance. It is easier to remember a constituent that occurred just recently in the sentence than one that occurred several embeddings ago. Interestingly, even though SPEC was especially designed to overcome such memory effects in the Parser's sequence memory, the same effect is generated by the Stack architecture. The latest embedding has noise added to it only once, whereas the earlier elements in the stack have been degraded multiple times. Therefore, the accuracy is a function of the number of pop operations instead of a function of the absolute level

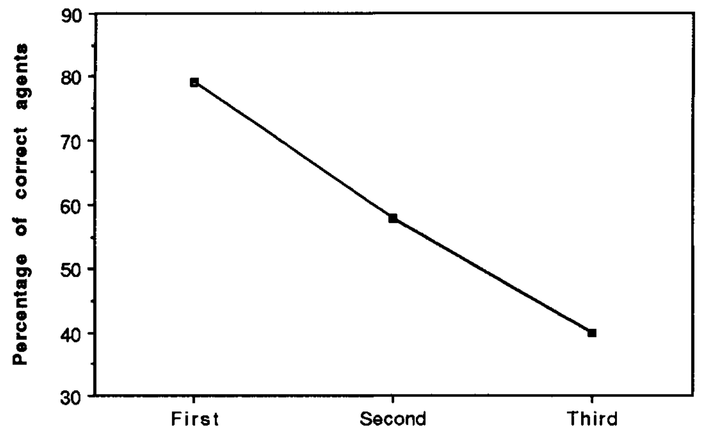


Figure 5 Memory accuracy after return from center embeddings (with 30% noise degradation). The percentage of correctly-remembered agents is plotted after the first, second, and the third pop in sentence templates 48 and 49 (represented by the words `boy`, `dog` and `girl` in the example sentences of table 3). Each successive pop is harder and harder to do correctly. Similarly, SPEC remembers about 84% of the patients correctly after the first pop, and 67% after the second pop.

of the embedding. With the example data, the percentage of correct agents after the first pop is always around 80%, whether that pop occurs after a single embedding (as in template 16), two embeddings (as in 40), or three (as in 48/49, figure 5).

When the SPEC output is analyzed word by word, several other interesting effects are revealed. Virtually in every case where SPEC made an error in popping an earlier agent or patient from the stack it confused it with another noun. In other words, SPEC performs plausible role bindings: even if the exact agent or patient is obscured in the memory, it "knows" that it has to be a noun. The weights of the Parser network have learned to encode this constraint. Moreover, SPEC does not generate the noun at random. Out of all nouns it output incorrectly, 75% had occurred earlier in the sentence, whereas a random choice would give only 54%. It seems that traces for the earlier nouns are discernible in the previous-hidden-layer pattern, and consequently, they are slightly favored at the output. Such priming effect is rather surprising, but it is very plausible in terms of human performance.

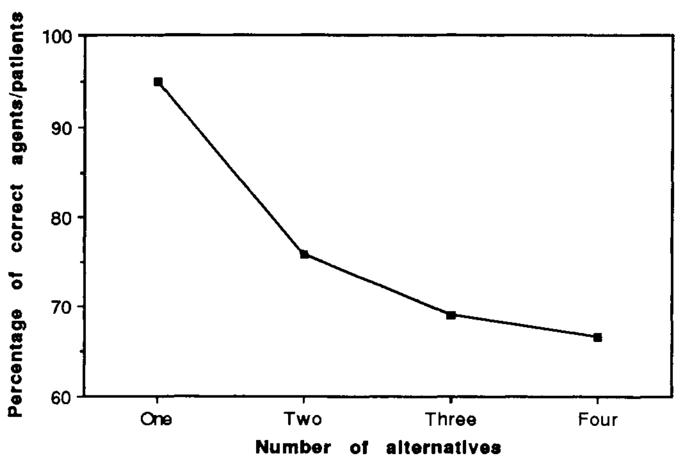


Figure 6 Effect of the semantic restrictions on the memory accuracy (with 30% noise degradation). The percentage of correctly-remembered agents and patients over the entire corpus is plotted against how strongly they were semantically associated with the verb. When there was only one alternative (such as `dog` as an agent for `bit` or `cat` as the patient of `chased`), SPEC remembered 95% of them correctly. There was a marked drop in accuracy with two, three and four alternatives.

The semantic constraints (table 2) also have a marked effect on the performance. If the agent or patient that needs to be popped from the stack is strongly correlated with the verb, it is easier for the network to remember it correctly (figure 6). The effect depends on the strength of the semantic coupling. For example, `girl` is easier to remember in `The girl, who the dog bit, liked the boy`, than in `The girl, who the dog bit, saw the boy`, which is in turn easier than `The girl, who the dog bit, chased the cat`. The reason is that there are only two possible agents for `liked`, whereas there are three for `saw` and four for `chased`.

A similar effect has been observed in human processing of relative clause structures. Huang [19] showed that young children understand embedded clauses better when the constituents are semantically strongly coupled. Caramazza and Zurif [6] observed similar behavior on aphasics. This effect is often attributed to impaired capability for processing syntax. The SPEC experiment indicates that it could be at least partly due to impaired memory as well. When the memory representation is impaired with noise, the Parser has to clean it up. In propagation through the Parser's weights, noise that does not coincide with the known alternatives cancels out. Apparently, when the verb is strongly correlated with some of the alternatives, more of the noise appears coincidental and is filtered out.

5 Discussion

SPEC is quite insensitive to configuration and simulation parameters. Many variations were tried in the experiments, such as hidden layers with 10–75 units, training sets with 200–4,000 sentences, different templates for training, modifying word representations in the Parser only, not modifying them at all, fixed learning rates 0.1–0.001 for weights and representations, gradually reducing the learning rates, training the modules together, and training them separately. All these variations led to roughly comparable results. Such flexibility suggests that the approach is very strong, and there should be plenty of room for adapting it to more challenging experiments.

Several other observations also indicate that the approach should scale up well. First, as long as SPEC can be trained with the basic constructs, it will generalize to a very large set of new combinations of these constructs. Combinatorial training [56] of structure is not necessary. In other words, SPEC is capable of *dynamic inferencing*, previously postulated as very difficult for subsymbolic systems to achieve [61]. Second, like most subsymbolic systems, SPEC does not need to be trained with a complete set of all combinations of constituents for the basic constructs; a representative sample, like the 200 out of 1088 possible training sentences above, is enough. Finally, with the FGREP mechanism it is possible to automatically form meaningful distributed representations for a large number of words, even to acquire them incrementally [32, 30], and the network will know how to process them in new situations.

The SPEC architecture was mostly motivated from the artificial intelligence point of view, that is, by the desire to build a system that (1) would be able to process nontrivial input like symbolic systems, and (2) makes use of the unique properties of distributed neural networks such as learning from examples, spontaneous generalization, robustness, context sensitivity, and integrating statistical evidence. While SPEC does not address several fundamental issues in connectionist natural language processing (such as processing exceptions and representing flexible structure), it goes a long way in showing that learning and applying grammatical structure for parsing is possible with pure distributed networks.

However, SPEC was not aimed at only generating best possible performance without an underlying Cognitive Science philosophy. The architecture is decidedly not a reimplementation of a symbol processor, or even a hybrid system consisting of subsymbolic components in an otherwise symbolic framework. SPEC aims to model biological information processing at a specific, uniform level of abstraction, namely that of distributed representation on modular networks. SPEC should be evaluated according to how well its behavior matches that produced by the brain at the cognitive level.

The most immediate direction for future work is to apply the SPEC architecture to a wider variety of grammatical constructs and to larger vocabularies. Two main issues need to be addressed in this work:

1. It will be necessary to develop methods for representing the final parse result. Currently, SPEC passes the output CRVs to an unspecified short-term memory system. This system needs to be made an explicit part of SPEC, preferably in such a way that the sentence representation can be used by other subsymbolic networks in processing multi-sentential text and in various reasoning tasks.

2. It might be possible to utilize the interpolation capability and context sensitivity of distributed neural networks at the level of processing structure. The current SPEC architecture generalizes to new instances of basic constructs, but generalization to new sentence structures is built in into the architecture. Perhaps a way can be found to generalize also at the level of control and segmentation. This way, the system could perform more robustly when the input is irregular (or ungrammatical), and contains novel basic constructs.

The Segmenter is perhaps the most significant new feature of the SPEC architecture. Most connectionist systems to date are based on simple propagation through homogenous networks or between networks of a modular system. As we have seen above, such systems are very good at dealing with regularities and integrating large amounts of small pieces of evidence, but they do not easily lend themselves to processing complex knowledge structures and unusual and novel situations. Such systems are not "conscious" of what they are doing, that is, they do not have representations concerning the nature of their internal representations and processes. As a result, they cannot employ high-level strategies in controlling the execution; their behavior is limited to a series of reflex responses.

With a comprehensive high-level monitor and control system, it would be possible to build much more powerful subsymbolic models. Current systems try to process every input in exactly the same way, regardless of whether the input makes sense or not. A high-level controller could monitor the feasibility of the task and the quality of the output, and initiate exception processing when the usual mechanisms fail. For example, unusual events or ungrammatical input could be detected and then processed by special mechanisms. The monitor could also clean up internal inaccuracies and keep the system execution on a stable path. Sequential high-level procedures and reasoning mechanisms could be implemented, such as comparing alternative interpretations and applying high-level rules to conclude new information. Equipped with such mechanisms, subsymbolic models would be able to perform much more robustly in the real world. Eventually, the goal would be to develop a distributed control system that would act as a high-level "conscious" monitor, similar to the central executive system in psychological and neuropsychological theories of controlled processes [3, 11, 27, 37, 40, 45, 48, 49, 51, 52].

The Segmenter is a first step toward implementing such a control system in the connectionist framework (see also [20, 21, 44, 60]). This module monitors the input sequence and the state of the parsing network, and issues I/O control signals for the Stack memory and the Parser itself at appropriate times. The Segmenter has a high-level view of the parsing process, and uses it to assign simpler tasks to the other modules. In that sense, the Segmenter implements a strategy for parsing sentences with relative clauses. Further developing such control mechanisms in parsing and in other cognitive tasks constitutes a most exciting direction for future research.

6 SUMMARY

SPEC is a distributed neural network architecture for parsing sentences with relative clauses. It receives a sequence of input word representations as its input, and generates a sequence of clause case-role representations as its output. SPEC consists of three modules: the Parser, the Segmenter, and the Stack. The Parser is a simple recurrent network that maps a sequence of words into a case-role representation. The Segmenter is a feedforward network that breaks the input sequence into clauses (so that the Parser only has to process one clause at a time) and controls the execution of the entire system. The Stack is a RAAM network that stores the state of the parse before a center embedding and restores it upon return from the center embedding.

By dividing the parsing task this way into transformation, segmentation and memory, the system only needs to be trained with the most basic relative clause constructs. During performance, SPEC generalizes not only to new instances of familiar sentence structures, but to novel structures as well, thereby demonstrating dynamic inferencing. Importantly, SPEC is not a neural network reimplementation of a symbol processor. It is purely a distributed system, and has the usual properties of such systems. SPEC exhibits plausible memory degradation as the depth of the center embeddings increases, and its performance is aided by semantic constraints between the constituents. SPEC is also self-contained: it controls its own execution during performance, and no external symbolic supervisor is needed. Scaling up the architecture towards processing real-world texts, designing a representation system for the parse result, and further expanding the mechanisms of connectionist high-level control constitute the main directions for further research.

ACKNOWLEDGEMENTS

Special thanks go to Dennis Bijwaard for the initial implementation and experiments that led to the SPEC architecture. Most of the simulations were run on the CRAY Y-MP 8/864 and the IBM RS6000s at the University of Texas Center for High-Performance Computing.

REFERENCES

[1] Robert B. Allen. Several studies on natural language and back-propagation. In *Proceedings of the IEEE First International Conference on Neural Networks* (San Diego, CA), volume II, pages 335–341, Piscataway, NJ, 1987. IEEE.

[2] Robert B. Allen and Mark E. Riecken. Reference in connectionist language users. In R. Pfeifer, Z. Schreter, F. Fogelman Soulié, and L. Steels, editors, *Connectionism in Perspective*, pages 301–308. Elsevier, New York, 1989.

[3] Alan D. Baddeley. *Working Memory.* Oxford University Press, Oxford, UK; New York, 1986.

[4] George Berg. A connectionist parser with recursive sentence structure and lexical disambiguation. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 32–37, Cambridge, MA, 1992. MIT Press.

[5] Douglas S. Blank, Lisa A. Meeden, and James B. Marshall. Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In John Dinsmore, editor, *The Symbolic and Connectionist Paradigms: Closing the Gap*, pages 113–148. Erlbaum, Hillsdale, NJ, 1992.

[6] Alfonso Caramazza and Edgar B. Zurif. Dissociation of algorithmic and heuristic processes in language comprehension: Evidence from aphasia. *Brain and Language*, 3:572–582, 1976.

[7] David J. Chalmers. Syntactic transformations on distributed representations. *Connection Science*, 2:53–62, 1990.

[8] Lonnie Chrisman. Learning recursive distributed representations for holistic computation. *Connection Science*, 3:345–366, 1992.

[9] Walter A. Cook. *Case Grammar Theory.* Georgetown University Press, Washington, DC, 1989.

[10] Cynthia Cosic and Paul Munro. Learning to represent and understand locative prepositional phrases. In *Proceedings of the 10th Annual Conference of the Cognitive Science Society*, pages 257–262, Hillsdale, NJ, 1988. Erlbaum.

[11] Nelson Cowan. Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological Bulletin*, 104:163–191, 1988.

[12] Charles Patrick Dolan. *Tensor Manipulation Networks: Connectionist and Symbolic Approaches to Comprehension, Learning and Planning.* PhD thesis, Computer Science Department, University of California, Los Angeles, 1989. Technical Report UCLA-AI-89-06.

[13] Michael G. Dyer. *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension.* MIT Press, Cambridge, MA, 1983.

[14] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[15] Jeffrey L. Elman. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning*, 7:195–225, 1991.

[16] Jeffrey L. Elman. Incremental learning, or The importance of starting small. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pages 443–448, Hillsdale, NJ, 1991. Erlbaum.

[17] Charles J. Fillmore. The case for case. In Emmon Bach and Robert T. Harms, editors, *Universals in Linguistic Theory*, pages 0–88. Holt, Rinehart and Winston, New York, 1968.

[18] Geoffrey E. Hinton. Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75, 1990.

[19] Ming S. Huang. A developmental study of children's comprehension of embedded sentences with and without semantic constraints. *Journal of Psychology*, 114:51–56, 1983.

[20] Robert A. Jacobs, Michael I. Jordan, and Andrew G. Barto. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15:219–250, 1991.

[21] Ajay N. Jain. Parsing complex sentences with structured connectionist networks. *Neural Computation*, 3:110–120, 1991.

[22] Janet L. Kolodner. *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* Erlbaum, Hillsdale, NJ, 1984.

[23] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[24] Michael Lebowitz. *Generalization and Memory in an Integrated Understanding System.* PhD thesis, Department of Computer Science, Yale University, New Haven, CT, 1980. Research Report 186.

[25] Geunbae Lee, Margot Flowers, and Michael G. Dyer. Learning distributed representations of conceptual knowledge and their application to script-based story processing. *Connection Science*, 2:313–346, 1990.

[26] Wendy G. Lehnert. *The Process of Question Answering*. Erlbaum, Hillsdale, NJ, 1978.

[27] Gordon D. Logan and William B. Cowan. On the ability to inhibit thought and action: A theory of an act of control. *Psychological Review*, 91:295–327, 1984.

[28] James L. McClelland and Alan H. Kawamoto. Mechanisms of sentence processing: Assigning roles to constituents. In James L. McClelland and David E. Rumelhart, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 2: Psychological and Biological Models*, pages 272-325. MIT Press, Cambridge, MA, 1986.

[29] Risto Miikkulainen. A PDP architecture for processing sentences with relative clauses. In Hans Karlgren, editor, *Proceedings of the 13th International Conference on Computational Linguistics*, pages 201-206, Helsinki, Finland, 1990. Yliopistopaino.

[30] Risto Miikkulainen. *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. MIT Press, Cambridge, MA, 1993.

[31] Risto Miikkulainen and Michael G. Dyer. Encoding input/output representations in connectionist cognitive systems. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 347-356, San Mateo, CA, 1989. Morgan Kaufmann.

[32] Risto Miikkulainen and Michael G. Dyer. Natural language processing with modular neural networks and distributed lexicon. *Cognitive Science*, 15:343-399, 1991.

[33] Paul Munro, Cynthia Cosic, and Mary Tabasko. A network for encoding, decoding and translating locative prepositions. *Connection Science*, 3:225-240, 1991.

[34] Allen Newell. Physical symbol systems. *Cognitive Science*, 4:135-183, 1980.

[35] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA, 1991.

[36] Allen Newell and Herbert A. Simon. GPS: A program that simulates human thought. In Edward A. Feigenbaum and Jerome A. Feldman, editors, *Computers and Thought*. McGraw-Hill, New York, 1963.

[37] Donald A. Norman and Tim Shallice. Attention to action: Willed and automatic control of behavior. Technical Report 99, Center for Human Information Processing, University of California, San Diego, 1980.

[38] Jordan B. Pollack. Cascaded back-propagation on dynamic connectionist networks. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, pages 391-404, Hillsdale, NJ, 1987. Erlbaum.

[39] Jordan B. Pollack. Recursive distributed representations. *Artificial Intelligence*, 46:77–105, 1990.

[40] Michael I. Posner and C. R. Snyder. Attention and cognitive control. In Robert L. Solso, editor, *Information Processing and Cognition*, pages 55–85. Erlbaum, Hillsdale, NJ, 1975.

[41] David E. Rumelhart, Geoffrey E. Hinton, and James L. McClelland. A general framework for parallel distributed processing. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 45–76. MIT Press, Cambridge, MA, 1986.

[42] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, Cambridge, MA, 1986.

[43] Roger C. Schank and Robert P. Abelson. *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*. Erlbaum, Hillsdale, NJ, 1977.

[44] Walter Schneider and Mark Detweiler. A connectionist/control architecture for working memory. In Gordon H. Bower, editor, *The Psychology of Learning and Motivation*, volume 21, pages 53–119. Academic Press, New York, 1987.

[45] Walter Schneider and Richard M. Shiffrin. Controlled and automatic human information processing I: Detection, search, and attention. *Psychological Review*, 84:1-66, 1977.

[46] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. Learning sequential structure in simple recurrent networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 643–652. Morgan Kaufmann, San Mateo, CA, 1989.

[47] David Servan-Schreiber, Axel Cleeremans, and James L. McClelland. Graded state machines: The representation of temporal contingencies in simple recurrent networks. *Machine Learning*, 7:161–194, 1991.

[48] Tim Shallice. Specific impairments of planning. *Philosophical Transactions of the Royal Society of London B*, 298:199-209, 1982.

[49] Tim Shallice. *From Neuropsychology to Mental Structure*. Cambridge University Press, Cambridge, UK, 1988.

[50] Noel E. Sharkey and Amanda J. C. Sharkey. A modular design for connectionist parsing. In Anton Nijholt Marc F. J. Drossaers, editor, *Twente Workshop on Language Technology 3: Connectionism and Natural Language Processing*, pages 87–96, Enschede, the Netherlands, 1992. Department of Computer Science, University of Twente.

[51] Richard M. Shiffrin and Walter Schneider. Controlled and automatic human information processing II: Perceptual learning, automatic attending, and a general theory. *Psychological Review*, 84:127–190, 1977.

[52] Richard M. Shiffrin and Walter Schneider. Automatic and controlled processing revisited. *Psychological Review*, 91:269–276, 1984.

[53] Robert. F. Simmons and Yeong-Ho Yu. Training a neural network to be a context-sensitive grammar. In *Proceedings of the Fifth Rocky Mountain Conference on Artificial Intelligence, Las Cruces, NM*, pages 251–256, 1990.

[54] Paul Smolensky. On the proper treatment of connectionism. *Behavioral and Brain Sciences*, 11:1–74, 1988.

[55] Paul Smolensky. Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216, 1990.

[56] Mark F. St. John. The story gestalt: A model of knowledge-intensive processes in text comprehension. *Cognitive Science*, 16:271–306, 1992.

[57] Mark F. St. John and James L. McClelland. Applying contextual constraints in sentence comprehension. In David S. Touretzky, Geoffrey E. Hinton, and Terrence J. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School*, pages 338–346, San Mateo, CA, 1989. Morgan Kaufmann.

[58] Mark F. St. John and James L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46:217–258, 1990.

[59] Andreas Stolcke. Learning feature-based semantics with simple recurrent networks. Technical Report TR-90-015, International Computer Science Institute, Berkeley, CA, 1990.

[60] Ronald A. Sumida. Dynamic inferencing in parallel distributed semantic networks. In *Proceedings of the 13th Annual Conference of the Cognitive Science Society*, pages 913–917, Hillsdale, NJ, 1991. Erlbaum.

[61] David S. Touretzky. Connectionism and compositional semantics. In John A. Barnden and Jordan B. Pollack, editors, *High-Level Connectionist Models*, volume 1 of *Advances in Connectionist and Neural Computation Theory*, Barnden, J. A., series editor, pages 17–31. Ablex, Norwood, NJ, 1991.

6

Towards Instructable Connectionist Systems

DAVID C. NOELLE
GARRISON W. COTTRELL

Department of Computer Science and Engineering
Institute for Neural Computation
University of California, San Diego
La Jolla, CA 92093-0114

1 INTRODUCTION

At least three disparate channels have been used to install new knowledge into artificial intelligence systems. The first of these is the *programmer channel*, through which the knowledge in the system is simply *edited* to include the desired new knowledge. While this method is often effective, it may not be as efficient as learning directly from environmental interaction. The second channel may be called the *linguistic channel*, through which knowledge is added by explicitly *telling* the system facts or commands encoded as strings of quasi-linguistic *instructions* in some appropriate form. Finally, there is, for want of a better phrase, the *learning channel*, through which the system learns new knowledge in an inductive way via environmental observations and simple feedback information. These latter two channels are the ones upon which we wish to focus, as they are the hallmarks of *instructable systems*. Most instructable systems depend upon, or at least heavily favor, one of these two channels for the bulk of their knowledge acquisition. Specifically, symbolic artificial intelligence systems have generally depended upon the explicit use of sentential logical expressions, rules, or productions for the transmission of new knowledge to the system. In contrast, many connectionist network models have relied solely on inductive generalization mechanisms for knowledge creation. There is no apparent reason to believe that this rough dichotomy of technique is necessary, however. Systems which are capable of receiving detailed instruction and also generalizing from experience are both possible and potentially very useful.

Consider an autonomous robot which is to operate in a typical office environment. It is reasonable to expect that some relevant knowledge concerning this environment could be extracted directly from the experience of the robot. Navigation about the work area, avoidance of obstacles, grasping of common objects, and similar skills could all be learned over time. However, it is unreasonable to expect the robot to use induction to operationalize advice like "it is best to remove the books from the shelves of a bookcase before attempting to move it". The robot must have some means of receiving such explicit instruction and operationalizing it. Instructability provides a way of overcoming gaps in knowledge overlooked by the programmer and of quickly acquiring knowledge yet to be learned by direct experience.

The integration of instruction and inductive learning may be approached from a number of directions. It may appear reasonable to begin with a system framework which uses a logic or some other sentential representation for knowledge. In such a framework, "learning by being told" comes almost for free. To the degree that a simple mapping exists between linguistic directives and quasi-linguistic knowledge objects, instruction becomes a simple matter of translation. Starting, then, from such an instructable framework, inductive generalization mechanisms may be added to the system. A good deal of research has been conducted on just this sort of integration [10] — on induction using a sentential knowledge representation. But there is another option.

A system which is meant to exploit both detailed instruction and experiential generalization may also be rooted in a learning framework, such as that of connectionist networks. In this case, inductive learning comes almost for free, since the wide array of robust connectionist learning techniques are available at the onset. The system design task then becomes one of finding a way to directly instruct such artificial neural networks without severely limiting their learning mechanisms. This is the approach to be discussed here.

This approach to instructable systems does not involve the use of a sentential knowledge representation but, like many connectionist designs, relies on the encoding of long term procedural knowledge as numerical weight values on connections between simple processing elements. Short term state knowledge can also be encoded numerically, as the activation levels of those processing elements. In other words, the knowledge that such an instructable system possesses about the problem that it must presently face is internally represented by a real *activation vector* of dimensionality equal to the number of processing elements in the system. The "state" of such a system at any point in time may be specified as one such activation vector, and a "computation" may be viewed as a dynamic trajectory through the corresponding vector space. Furthermore, the real vector knowledge encodings of such connectionist systems do not lend themselves to semantic interpretations based on the simple decomposition of concepts into primitive features corresponding to the ordinal dimensions (i.e., the processing elements) of the activation vector space. The knowledge representations to be used in this approach are *distributed*, which is to say that the activity level of any individual processing element need not carry any independent meaning. The state knowledge of these systems is encoded across whole activation vectors and may not be trivially mapped into a compositional semantics.

Such a distributed knowledge representation may seem like a handicap to an instructable connectionist system which is expected to understand and operationalize linguistic advice, but there are several key advantages to this representational strategy. These advantages include the availability of powerful learning algorithms, the efficient use of representational resources, and the enhancement of fault tolerance.

First, and foremost, of these advantages is the access to powerful connectionist learning algorithms afforded by this choice of representation. Many training techniques for multiple layer connectionist networks, such as the well studied backpropagation learning algorithm [11], tend to produce systems which utilize distributed representations over their processing elements. Accepting such representations opens the door to use of these powerful layered network learning methods. Also, this focus on experiential learning lends an air of cognitive plausibility to this strategy in that representational features which appear to be learned in humans may also be learned by a multiple layer connectionist system [8].

Another advantage of such learned distributed representations involves their relative flexibility and efficiency in distributing representational resources (i.e., the "meaning" of processing element activity) in a manner appropriate to the behavioral tasks at hand. Unlike a representational strategy which assigns an *a priori* meaning to the activation level of each processing element, a distributed representation approach may semantically "carve up" the activation vector space in any way which facilitates proper system response. For example, features which require fine grained discrimination may be represented over a large region of the activation vector space, utilizing a significant fraction of the available representational resources, while less important features may be represented over smaller regions. In this way, a learned distributed representation may be automatically tailored to the domain of the system.

As an added bonus, this efficient distribution of representational resources may also result in encodings which are robust in the face of noise and loss of processing elements. Distributing represented information across a sufficiently large set of processing elements often results in representations which degrade gracefully when damaged. This automatic utilization of excess resources to enhance fault tolerance is yet another benefit of adopting a system design strategy which involves the induction of distributed representations.

In short, representing knowledge as distributed activation space vectors strongly supports the inductive learning aspect of instructable systems. What is needed is an approach to incorporating linguistic advice into such a connectionist learning model. Such integration is the focus of this chapter.

The basic idea of our approach is taken from an observation by Paul Churchland [1]. Clearly, standard weight change mechanisms are too slow to account for "learning by being told". Instead, the proper mechanism for instruction based learning is changes in *activation space*. Shifts in activation occur at the right time scale for such learning. However, for such a learning mechanism to work, we need an activation space that consists of *articulated attractors*. In other words, for a connectionist network to receive propositional knowledge via instruction, the activation space of that network should give rise to a combinatorial number of possibilities for attractors, corresponding to the propositions that may be represented. This may be envisioned as a series of attractors for the arguments of each proposition and for each predicate. This slavish following of logical terminology is not to be construed to mean that we are proposing yet another connectionist implementation of predicate logic! Rather, these terms provide the simplest way to describe what we hope will be a much richer formalism. For example, one obvious advantage to this kind of approach is that *constraints* between arguments and the predicate can easily be encoded in connections between these attractor bowls. Distributed representations of predicates, and thus, complicated carvings of the attractor bowl surface in response to shades of meaning, are another consequence of this view.

In this way, knowledge provided through the linguistic channel may be stored, at least temporarily, in the activation space of the system. Longer term retention of this knowledge may be had by allowing weight changes to deepen the attractor wells, perhaps through Hebbian learning, so that a given pattern in the activation space will become more likely to be reinstated in the future. An important point to take away is that *the system can only "learn" by this mechanism what it almost already knows*. What we mean by this is that the attractor space, or the activation dynamics, must contain attractor wells or activation trajectories that are easily instantiated, and meaningful to the system as such. The system may never have actually visited a given region of activation space before being presented with some instruction, but the appropriate attractor in this region must be relatively stable nonetheless.

Finally, it should not be forgotten that the primary goal of "learning by being told" is improved system performance. An instructable system must be able to demonstrate, operationally, that it has acquired the knowledge that we have endeavored to impart. We take this to be a definitional aspect of "learning by being told".

To summarize, an instructable system must demonstrate the three R's: *represent*, *remember*, and *respond*. If the system cannot *represent* the results of linguistic input and *remember* it, the system will not "learn by being told". If it does not *respond* appropriately after instruction, it has not learned from that instruction. These are rather obvious points to anyone who has studied psychology, but it is important for connectionists to be explicit about the possible mechanisms that may underlie such transformations. The issue of "learning by being told" cannot be avoided. It must be addressed if connectionist approaches to cognition are to survive.

Connectionist "learning by being told" may be accomplished in a number of different ways, depending on the domain. For example, a question-answering system, after being told some facts, should be able to correctly answer questions concerning those facts, whether by rote, or preferably, by demonstrating inference from the given facts. This would typically require a system that understands both statements and questions on the same sensory channel, and is also capable of generating linguistic strings as answers.

Another version of this idea would be to have a two-part system in which one part, which performs a given task, is manipulated by another part which uses advice or instruction to alter what the first part does. In this chapter, we will focus on this approach. Specifically, we will give a mapping network a series of instructions on how to map inputs to outputs. Here we are most concerned with demonstrating "proof of concept". While this demonstration system does display systematic responses to our instruction, we have not implemented the storage of these instructions as attractors. This is a small extension to what we have done. This example system may also be criticized as being rather small. In short, the models presented here should be seen as working "*towards* instructable connectionist systems", and it should be recognized that there is still a long way to go.

To motivate our ideas, we first review two independent previous systems. The first is a network capable of learning simple symbolic arithmetic procedures for arbitrarily sized numeric inputs. This system demonstrates a useful strategy for generating connectionist systems capable of systematic behavior. The success of this network also shows that a learned distributed representation may be effectively used to perform a discrete symbol manipulation task. Next, a connectionist model is presented which learns to map complex sensory events into temporal pattern sequences which represent descriptive linguistic strings. This model demonstrates how temporal sequences of activation patterns may be used to represent linguistic statements and how such sequences may come to be naturally associated with the elements of some task domain. This system is also used to show how distributed representational resources may be automatically allocated in an efficient and appropriate manner.

With the techniques exemplified by these two systems in hand, some initial attempts towards the formulation of a connectionist model of "learning by being told" will be discussed. Our demonstration system allows for the performance of a systematic symbol manipulation task, modulated by instructions encoded as temporal sequences of activation patterns. The modulation of behavior is accomplished by allowing input instruction sequences to prejudice the activity of a domain task network towards appropriate regions of that network's activation space. The results of experiments involving a simple symbolic mapping task are presented, and the application of this strategy to the problem of symbolic arithmetic is briefly discussed.

2 SYSTEMATIC ACTION

2.1 CONNECTIONIST SYSTEMATICITY

In this section we review a simple model of procedural learning by Cottrell & Tsung [3]. This model (among many others) demonstrates that systematic behavior is possible in connectionist networks. Some critics of connectionism have posited that connectionist networks are incapable of certain systematic reasoning tasks without first *implementing* some sort of symbolic architecture [6]. It is, therefore, important for the justification of the instructable network approach being explored here to demonstrate that such networks can indeed perform systematic symbol manipulation tasks while retaining their important inductive learning mechanisms.

The systematic symbol manipulation task of interest here is simple addition of arbitrarily sized natural numbers. The numbers are represented as strings of digits in some radix, and the two strings are assumed to be aligned by column, so two digits of the same magnitude (i.e., in the same "place") are adjacent. Such an arithmetic task may be accomplished by a connectionist network by presenting the network with one column of digits at a time. The system is then to systematically step through the actions of addition: announcing the sum of the two visible digits, announcing when a carry is generated by the addition, and requesting the next column of digits. Such systematic behavior may be used to demonstrate that connectionist networks are capable of learning behaviors analogous to simple programming constructs, such as action sequences, conditional branches, and condition bounded iteration. Such a network shows that connectionist systems are capable of systematic symbol manipulation tasks.

2.2 ARITHMETIC

Rumelhart et al. [8] proposed that a symbol processing task, such as arithmetic addition, may be accomplished by a connectionist system by first creating a "physical representation" of the problem, and then modifying this representation by means of a pattern association process. For example, the problem of adding "327" and "865" may be encoded as a "physical representation" that looks like:

```
327
865
---
```

A system which is given this representation (or some appropriate portion of it) as input is now faced with a pattern recognition problem. A connectionist network may be trained to respond to this input representation by recording a sum and marking a carry, producing the new physical representation:

```
 1
327
865
---
  2
```

This new representation may be iteratively presented to the network to produce the sum value for the next column. This process may continue until the arithmetic operation is complete.

In hopes of handling arbitrarily large numbers, the addition problem of interest here will assume some sort of rudimentary attentional process which may segment the physical representation into "interesting" portions. Specifically, at any given point in time, only one column of digits will be made visible to the system, starting with the rightmost column. The system will be expected to issue a particular output signal in order to have the next column presented. This formulation of the problem allows a network with a fixed size input to handle arbitrarily long digit sequences, and it also forces a successful network to exhibit some degree of systematicity in its behavior.

Also, the system at hand will not be allowed to mark the "physical representation" with carry information from one column to the next. The inputs to the system will *not* include information concerning the presense of a carry in a given column. The network will be forced to *remember* the state of a carry bit as consecutive columns are processed. This means that the system must retain some "internal state" over time.

Formally, the adder network is to be presented with an input pattern vector consisting of three parts: the two digits in the current column, and a single bit flag which signals the end of the digit strings to be added. The system has four exclusive output signals which it may generate, encoded in a "1-out-of-N" localist fashion: WRITE, CARRY, NEXT, and DONE. Another portion of the network output is dedicated to an encoding of a sum digit to be written in the result field of the current column. The system is expected to sequentially process the columns of digits according to the following algorithm:

```
while ( not done ) do
    begin
        output (WRITE, low order digit of sum)
        if ( $sum \geq radix$ ) then
            output (CARRY, anything)
        output (NEXT, anything)
    end
if ( there was a carry on the last column ) then
    output (WRITE, "1")
output (DONE, anything)
```

Time Step	Done?	1st Digit	2nd Digit	Action	Result
1	No	7	5	WRITE	2
2	No	7	5	CARRY	—
3	No	7	5	NEXT	—
4	No	2	6	WRITE	9
5	No	2	6	NEXT	—
6	No	3	8	WRITE	1
7	No	3	8	CARRY	—
8	No	3	8	NEXT	—
9	Yes	0	0	WRITE	1
10	Yes	0	0	DONE	—

Table 1 An Example Addition: $327 + 865$

One time step is to pass for each "**output**" command which is executed in this algorithm. New inputs (i.e., the next column of digits) are presented to the network on the time step which immediately follows any time step in which the NEXT output is issued. Note that there is no explicit representation of the carry bit amongst the inputs made available to this program at any given time step. The system is expected to *remember* carry information between iterations of the loop, and implicitly apply this information to the computation of the next sum and the next carry. An example execution of this algorithm is presented in Table 1.

2.3 The Model

A simple feed-forward network is not sufficient for this addition problem. For a connectionist system to generate temporal sequences of actions, as is expected here, it must have some way to remember "where it is" in the current sequence. A typical connectionist network's only form of short-term memory is in the form of internal activation states, so any network which is to perform action sequences must have access to different activation states at different points in the desired sequences. One way to allow such a network to have distinct activation states is to introduce *recurrence* into the network. This essentially involves providing as input to a network the previous activation state of some portion of the network's processing elements. Access to this

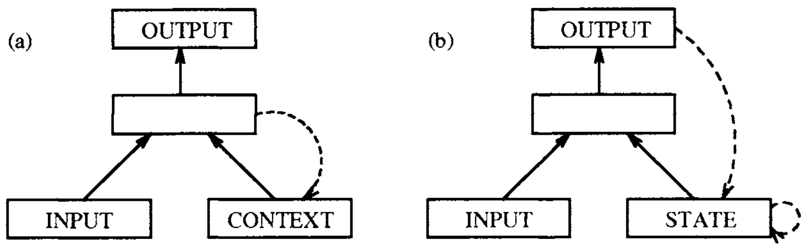


Figure 1 Network Architectures — (a) simple recurrent network; (b) Jordan network

previous activation information allows a network to determine its location in a sequence of actions, making the generation of such sequences possible. Recurrence was used to maintain activation state in the model presented here.

Note also that this adder system is expected to *remember* the value of a carry bit over multiple time steps. This means that the adder network must encode carry information into the activation state of the processing elements which provide recurrent inputs. By performing such an encoding, the system will be able to modify its behavior based on the activation levels of these recurrent inputs, thereby *recalling* and utilizing the required carry information.

Two basic recurrent network architectures were examined for use as adders. Both of these architectures are shown in Figure 1. Each box in Figure 1 represents a layer of connectionist processing elements, and each solid arrow represents full interconnection between the two given layers. The dashed arrows correspond to fixed, one-to-one connections between layers. In both architectures, the *input* layer consisted of a single binary "done" flag and two binary encoded input digits. In all of these experiments a radix of 4 was used for the numbers to be added, so two processing elements were used to encode each digit. The *output* layer of both network types included four elements to provide a localist code for the four possible actions — WRITE, CARRY, NEXT, and DONE — and two more elements for a binary encoded result digit. Various hidden layer sizes were examined, with the results reported here originating from a network with 16 hidden units. The first of these architectures is Elman's *simple recurrent network*, in which the hidden layer provides recurrent inputs to the network [5]. As shown in the figure, this recurrence is implemented by "unrolling the network in time" [11] for a single time step — essentially recording previous hidden layer activation levels in a special "context" layer which provides input on the following time step. This is equivalent to complete interconnectivity between the processing elements at the hidden layer. The second architecture is based on the work of Jordan, and it provides recurrence from the output layer rather than from the hidden layer [7]. This is implemented by maintaining a special "state" layer with a decaying average of output layer processing element activation levels. In other words, the activation level of the "state" layer at some time, t, is the sum of the previous, time $(t-1)$, "state" activation, attenuated by some proportion, μ, and the activation of the output layer at time $(t-1)$. Both of these architectures provide the recurrence which is needed to enact sequential behaviors, and both may be trained using simple variations of the backpropagation learning algorithm [11].

2.4 The Training Procedure

Both connectionist network architectures were taught to execute the addition algorithm by repeated exposure to a training corpus of base 4 addition problems. The problems in this corpus were restricted to contain numbers with no more than 3 digits. Despite this restricted problem size, the resulting training set was extremely large, and training times on the entire corpus turned out to be prohibitive. Training sets consisting of small random subsets of the whole training corpus were used in an attempt to decrease training times, but these trials resulted in networks which essentially "memorized" the given training sets and exhibited poor generalization. Clearly some non-standard training regime was needed to prepare the adder networks properly.

The training technique which was invented to solve this problem was dubbed *combined subset training*. This method involves training on a small random subset of the entire training corpus initially, and then successively doubling the training set size until success on the entire corpus is achieved. Training is terminated either when the entire corpus is presented for training and successfully learned or when the network successfully generalizes to all of the remaining cases in the training corpus. *Combined subset training* was used throughout the adder network experiments described here.

2.5 The Results

The *combined subset training* procedure allowed both simple recurrent networks and Jordan networks to successfully learn the addition algorithm. Indeed, after being exposed to only 8% of the entire training corpus, the networks were found to generalize well to all of the remaining number pairs. Further tests were conducted to examine how well the networks generalized to numbers with more than 3 digits. The networks were tested on random sets, consisting of 10 problems each, involving numbers with up to 14 digits. They were found to miss only one addition out of ten on average. In hopes of improving performance on the longer digit strings even further, a special "clean-up" training procedure was tried. The networks were subjected to a series of training sessions, with each session consisting of 10 epochs using the set of 10 problems which exhibited the most error. After 40 epochs of such training, the networks successfully generalized to arbitrary addition problems containing numbers with up to 14 digits!

Since both simple recurrent networks and Jordan networks generally behaved similarly in these experiments, some attempts were made to find differences between them with regard to their use in performing systematic tasks such as addition. Both architectures were trained to emulate an alternative addition algorithm, one which performed its output operations in a slightly different order than the original algorithm. This new operation permutation delayed the announcement of a carry until *after* the sum digit for the next column had been written. This new algorithm resulted in a problem which was inherently unsolvable by a Jordan network architecture. Thus, a subtle relationship was shown between choice of network architecture and choice of symbolic representation for a given problem.

Lastly, the learned internal representation of "state" information was investigated by a careful examination of hidden layer activation vectors over time. The trajectory of the hidden layer states through activation space was tracked in hopes of uncovering the internal representations of the learned programming constructs. Briefly, the hidden layer activation vectors of a simple recurrent network were recorded over the course of 10 addition problems, and a principal components analysis was performed on the resulting set of vectors. The activation vectors were then projected into the plane defined by the first two principal components, thereby generating a two dimensional plot of the hidden layer trajectories, focusing on the activation space dimensions of highest variance. A cartoon summary of such a plot is shown in Figure 2. Each point in the original diagram fell into one of the regions shown in the figure. The regions are labeled by the output action of the network corresponding to the given hidden layer activation. The arrows show the state transitions of the network between these regions. Thus, the network was shown to have implemented a kind of finite state automata. It is interesting to note that an output action (NEXT) may correspond to two different regions in the underlying state space. This was necessary because carry information had to be maintained by the network while it generated a NEXT action so that the correct sum could be

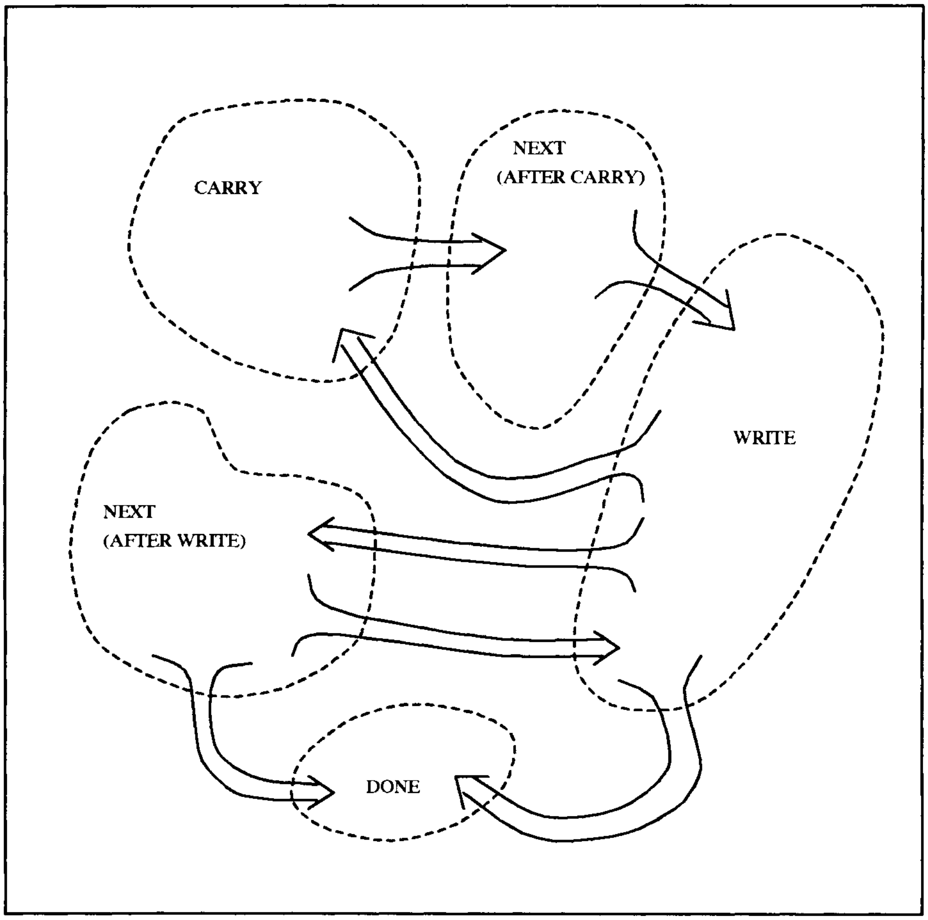


Figure 2 Cartoon Of State Space Plot

computed in the next time step. This carry information was plainly recorded in the location of the hidden layer activation state during each NEXT output. This analysis of hidden layer trajectories indicates that the connectionist inductive learning process successfully partitioned activation space in a manner which allowed for the recording of needed "state" information and for the appropriate systematic response to such information.

2.6 THE POSSIBILITIES

These addition networks clearly demonstrate that recurrent backpropagation networks may be trained, via an inductive learning mechanism, to perform systematic symbol manipulation tasks. Some limitations of such an approach should be noted, however. Since "internal state" in these networks is encoded in a fixed size vector of activation values, it is reasonable to expect that there will be severe limits on the length of time over which a bit of state information may be maintained. Similarly, there should be limits on the "depth of embedding" of implemented control constructs, such as the nesting depth of **while** loops. Note also, that the success of this system depended upon the accurate segmentation of the domain task into sequential chunks. It would have been unreasonable to expect the network to *remember* all of the input digits and perform the addition "in its head". In short, a more efficient connectionist model of memory is needed to extend this network design strategy into the realm of complex cognitive tasks.

Despite these limitations, the presented adder networks must be seen as strong indications of the power of recurrent connectionist networks to perform systematic tasks. It may even be possible to construct basic reasoning systems using this kind of architecture by training "rules" into the associational mapping behavior of a network and recording intermediate conclusions in the activation state of recurrent layers.

3 LINGUISTIC INTERACTION

3.1 LINGUISTIC SEQUENCES

In addition to a mechanism for the generation of systematic action, an instructable connectionist system requires some means of receiving quasi-linguistic input sequences and some means of translating these into some modulating force on the system's behavior. The solution proposed here involves *training* connectionist networks to modify their own behaviors based on input time-varying streams of quasi-linguistic tokens. Networks are expected to learn the meanings of linguistic sequences in terms of the elements of some domain task. Domain task experience coupled with exposure to relevant linguistic descriptions should allow connectionist systems to associate linguistic tokens and constructs with corresponding behavior. In this way, artificial neural networks may *ground* linguistic meaning in *perception* and *action*. In this view, linguistic statements become temporal patterns which appropriately trained networks may map into useful internal representations in the form of hidden layer activation levels.

There have been a number of connectionist systems which have utilized this view of linguistic sequences as activation patterns [5] [12],but only one such system will be described here. The task of interest here is a *description* task which involves the generation of a stream of linguistic tokens which accurately describe a perceived time-spanning event. In essence, this is the opposite of "learning by being told". It requires that quasi-linguistic sequences be generated from the activity of some complex *perception* subsystem, as opposed to the instructable network task of modulating activity in a complex *action* subsystem based on perceived quasi-linguistic input. Still, the basic mechanisms for representing linguistic statements as temporal activation patterns and for grounding these statements in a domain will prove to be essentially identical across these two system tasks. In order to demonstrate the viability of this scheme, a brief overview will be provided here of a connectionist network which is capable of performing the description task of providing "subcaptions" to simplified "movies". This network was generated and examined by Cottrell, Bartell, and Haupt [2].

3.2 SUBCAPTIONS

The problem of interest here involves generating a string of quasi-linguistic tokens which may be reasonably interpreted as a description of some perceived event. In order to keep the problem simple, the set of possible events shall be quite constrained. A small visual field, 4 pixels in width and 2 in height, is to be presented to the system as an 8 element bipolar vector. Only one object may appear in this visual field at any given time — a single pixel sized *ball*, and the location of this ball is specified by an input of 1.0 at the appropriate pixel. Empty pixels are specified with a -1.0 input value. A temporal sequence of visual scenes (i.e., a "movie") is to be presented to the system, depicting the ball *rolling*, *flying*, or *bouncing* either from left to right or from right to left. An event consists of four such scenes, presented contiguously and delimited by one to three scenes in which the ball does not appear at all. The job of the system is to receive this temporal sequence of visual images and produce a linguistic *description* of the perceived event. Descriptions are to take the form of temporal sequences consisting of three linguistic tokens: a noun, a verb, and an adverb. The only noun needed is "ball". Verbs are to include "rolls", "flies", and "bounces", and the possible adverbs are "left" and "right". The system is expected to output one linguistic token per time step for three consecutive time steps, producing descriptions such as "ball bounces right".

Movies are to be presented in a continual stream to the system, delimited only by short periods in which the visual field is empty. At any point in time it should be possible to freeze the current movie and have the system produce a description of it. This means that the system must essentially *forget* about old movies and must focus on quickly identifying the nature of the current event. Information concerning the *type* and *direction* of ball motion must be quickly extracted from the scenes and must be represented internally in a manner which will facilitate the generation of the appropriate linguistic description.

3.3 THE MODEL

As in the adder networks, recurrence is needed in order to solve this problem. To recognize a pattern of ball motion the system must be able to *remember* information concerning previous locations of the ball. Furthermore, in order to generate linguistic sequences over time the network needs to constantly keep track of "where it is" in the current output sequence. As was demonstrated by the adder networks, recurrent backpropagation networks are quite capable of learning to record this kind of "state" information in the activation levels of their recurrent layers.

With this in mind, the *Movie Description Network* was constructed by connecting two of Elman's simple recurrent networks [5]. One simple recurrent network was used to track ball motion in the input images, and another was used to step through the linguistic token sequences of output descriptions. The architecture of this network is show in Figure 3. The network was operated by presenting one movie image at the *scene* input layer at every time step, and propagating the resulting activation at least as far as the *gestalt* layer. It was hoped that the activation levels of processing elements in the *gestalt* layer would encode all of the information needed to generate an accurate description

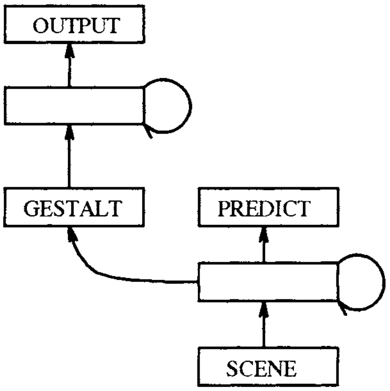


Figure 3 The Movie Description Network

of the current movie.[1] When such a description was desired, the activation states of layers in the movie perception subnetwork were "frozen", and the current state of the *gestalt* layer was used as a constant input to the description generation subnetwork. This description subnetwork was then "reset" (i.e., the activation levels of all of the processing elements in its recurrent hidden layer were set to an average value — namely, zero) and clocked through three time steps to allow it to generate a quasi-linguistic event description at its *output* layer. After a description had been produced, the perception subnetwork could continue to receive movie images, and further descriptions could be elicited.

Many experiments were conducted using this network architecture, using various network layer sizes and training parameters. The experiments described here used a network with an input *scene* layer size of 8 processing elements and a linguistic token *output* layer also of size 8. The six possible output tokens were sparse coded over these 8 outputs. The *gestalt* layer in these experiments contained 8 processing elements, as well. The hidden layer of the perception subnetwork was set to contain 12 elements, and the hidden layer of the description subnetwork was set to contain 20. All processing element activation levels were bounded by a sigmoid, forcing outputs to the range from -1.0 to 1.0.

[1]The name of the *gestalt* layer comes from the *Sentence Gestalt* network of St. John and McClelland, which had a major influence on this design. The *Sentence Gestalt* network has been successfully used to perform highly complex natural language question answering tasks [12].

3.4 THE TRAINING PROCEDURE

Like the adder networks, and simple recurrent networks in general, the *Movie Description Network* could be trained by a variation of the backpropagation learning algorithm using only a supervised error signal at the final *output* layer. Instead, this network received error information from two different sources, and it combined and applied this error information using standard backpropagation techniques. A supervised training signal was indeed provided at the linguistic *output* layer, specifying the correct sequence of tokens to describe the current event, but another signal was provided exclusively to the perception subnetwork at the *predict* layer. This layer was of the same size as the input *scene* layer and was intended to create an output image identical to the *next* image to be perceived. In other words, the *predict* layer was meant to be a prediction of the next "frame" in the current movie. This means that the perception subnetwork was shouldered with the task of understanding potential ball motion well enough to predict the ball's trajectory across the field of view. The error signal which was provided at the *predict* layer was simply generated from the actual scene that was observed at the next time step. In essence, this was an *unsupervised* (or *self-supervised*) training signal, since no information beyond the normal input sequence of images was needed to generate it. The *Movie Description Network*, therefore, used backpropagation to combine a supervised error signal at the linguistic *output* layer with an unsupervised error signal at the *predict* layer to produce weight modifications which drove the network both to make predictions about the current movie and to linguistically describe it.

Note that this dual feedback training technique placed the two subnetworks in competition over the representational resources of the perception subnetwork's hidden layer. If the perception subnetwork had been trained alone, the information encoded at this hidden layer would have included only items relevant to the prediction task. This need not include all of the information required for the description task. For example, when the ball was moving to the right and was currently in the rightmost column of the field of view, no information concerning the "type" of motion of the ball was needed to perform the prediction task. The next image was always the empty scene. The description network, however, still needed to know whether the ball was flying, rolling, or bouncing. Conversely, if no feedback had been provided at the *predict* layer, the perception subnetwork's hidden layer would have been devoted to the description task. No distinction between a bouncing ball that entered the field of view low and a bouncing ball that entered high would have needed to have been made, for instance. Such a distinction was critical for the prediction problem, however. In short, providing both sources of error feedback pushed the network to record a wider variety of properties of the input movies, and, thereby, to have a richer internal representation of the perceived events.

The exact training regime for the *Movie Description Network* involved repeated exposure of the network to movies and their corresponding quasi-linguistic descriptions. Two different methods were used for selecting the next movie to be presented to the network. First, a deterministic method was tried in which all of the movies in a training corpus were presented repeatedly in a fixed order. Second, a random method in which the next movie was repeatedly selected uniformly at random from the entire training corpus was applied. In the deterministic case, the network was trained for 8000 epochs, where an epoch was a complete pass through the training corpus. The random method was applied for 1.6 million individual time steps. In both cases, training progressed by first presenting an input image to the network and then training the perception subnetwork using prediction feedback. Then, before the next scene was input, both subnetworks were trained using error feedback from the entire movie description. In other words, a complete description was solicited from the description subnetwork in between image presentations, and error on this description was propagated back through the entire network.[2] Once feedback was provided on the movie description, the next image was presented to the network and the training procedure iterated. Throughout this process a learning rate of 0.01 was used, along with a momentum value of 0.9. When the maximum number of training epochs (or time steps) was reached, the error history of the learning network was examined. The epoch at which the network achieved a minimum average sum-squared error was identified, and the network's weights were reset to their values at that epoch. The end result of this training procedure was a network capable of both predicting ball trajectories and generating event descriptions.

3.5 The Results

The trained network proved successful at both the prediction and description tasks. When too little information was provided to identify an event, as was the case during the very first frame of each movie, the network failed at both tasks, as should be expected. Prediction average sum-squared error rates were on the order of 0.24 at these times. However, whenever sufficient movie segments

[2] Actually, when the random movie selection method was used, training on the output description was also done randomly. Between any two scene presentations there was a 50% chance of eliciting a description from the network and providing error feedback on it.

were presented to the network, error dropped below 0.001. The network correctly predicted ball trajectories and correctly described each event.

An analysis of the internal representations formed at the *gestalt* layer and at the perception network's hidden layer was conducted. The findings of this examination supported earlier intuitions concerning the competing requirements of the prediction and description tasks. For example, the internal representations of two distinct events were observed — of a ball bouncing right and entering the view low, and of a ball bouncing right and entering the view high. At the *gestalt* layer these events provoked almost identical representations as activation vectors. This is sensible since both events share the same description: "ball bounces right". At the hidden layer of the perception network, however, the two events generated widely different representations (as measured by Euclidean distance). This, too, is reasonable since the two events imply essentially opposite predictions at the *predict* layer. While the hidden layer clearly encoded all of the information needed to perform both tasks, it appears as if the *gestalt* layer was used by the network to cluster internal representations of events in a manner which facilitated the generation of quasi-linguistic descriptions.

Some final experiments were conducted which involved training the *Movie Description Network* without use of the unsupervised prediction error signal. It was found that this signal was *not* needed to achieve good performance on the description task. While including the prediction feedback increased the amount of information contained in the internal representations of events, such a rich representation did not significantly affect, for better or worse, the ability of the network to make the needed linguistic discriminations.

3.6 THE POSSIBILITIES

The success of the *Movie Description Network* at its description task demonstrates how recurrent connectionist networks may process linguistic information. The inductive learning mechanisms of such networks allow them to generate their own internal representations of perceptions, actions, linguistic tokens, and the relationships between these things. These internal representations are formed to facilitate the tasks on hand. When linguistic statements are encoded as temporal activation patterns, connectionist systems, such as this subcaptioning network, may be trained to generate statements appropriate to a given domain task. As the following explorations shall show, such encodings may also make instructable artificial neural networks possible.

4 Learning By Instruction

4.1 A Synthesis

The connectionist adders demonstrated that connectionist systems may perform systematic symbol manipulation tasks. The *Movie Description Network* showed how temporal activation patterns may be used to represent linguistic strings and how such strings may be grounded in the elements of some domain task. These two strategies may now be combined to produce a connectionist network that is capable of responding appropriately to some simple instruction sequences.

How should such an instructable network be designed? The only knowledge available to a typical connectionist system is in the form of connection weights and vectors of processing element activation levels, so any mechanism for instruction following must make modifications to one of these two sets of system variables. Some attempts to formulate techniques for the partial programming of connectionist networks have involved approaches which *compile* symbolic rules directly into connections and connection weights [4]. In some of these techniques the weights are later refined by standard connectionist inductive learning processes [13]. A major drawback of these "weight specification" approaches is that advice may only be given *before* training begins. Standard connectionist learning methods generally change the representational nature of weight values (i.e., how they relate to desired behaviors and entities *in the world*) in hard to predict ways, making the direct manipulation of those weights in response to instruction quite problematic. Prohibiting all instruction once inductive learning has begun is both cognitively unrealistic and potentially troublesome for practical systems. If the desired system is to be instructed in the midst of performance or if it is to learn continuously while doing its job, then the strategy of encoding rules as initial weights will not work. There is another option, however. Instructions may be encoded as activation patterns, and connectionist networks may be *trained* to respond to certain patterns of processing element activity as advice.

The idea is to encode instructions as network input vectors and to teach the system to respond to those instructions appropriately. In short, the instructable network has its set of inputs divided into two classes: domain task related inputs and instruction inputs. The domain inputs are used to communicate task parameters to the system, such as a column of digits in the adder networks. The instruction inputs receive vector encoded quasi-linguistic tokens which are to be interpreted as advice. The system produces output values which are contingent on the domain inputs, and this mapping process is modulated by the sequence of instruction inputs which are presented. Some error correction (supervised) training is used to get the network to "understand" the instruction token encodings in terms of the domain task, but once this training is completed the system may respond to advice *immediately*, without further connection weight modification. Alternatively, weight modification using a standard connectionist learning law may continue after the receipt of advice, with input instructions acting in a supplementary role.

An extremely simple version of this approach could involve encoding entire collections of advice as fixed sized real vectors which are provided as input to a simple feed-forward backpropagation network [11]. The domain problem of mapping domain task input vectors to some set of output vectors may be modulated in this network by the specification of an appropriate instruction input vector. This system design strategy transforms the problem of learning from instruction into a straightforward mapping problem.

While the use of a fixed size advice input vector is all that is really needed to get a feed-forward connectionist network to take advice, the encoding of whole collections of instructions as such fixed size vectors is a somewhat awkward process. It is easy to design a simple encoding which represents advice as a fixed size vector, but such fixed size representations generally place a hard limit on the size of the encoded instruction collection. A more natural way to present advice to the system is as a temporal sequence of symbolic tokens which form sentential recommendations, such as "$rock \Rightarrow paper$" to encode, "if the input is *rock* then specify *paper* as the output". Ideally, we would like to have a connectionist system which can handle arbitrarily long sequences of such advice tokens.[3]

This is where the strategy which was exemplified by the *Movie Description Network* may be applied to the instructable network problem. A temporal sequence of input activation vectors encoding instructional tokens may be used to present advice to the system, and the system may be trained using a standard connectionist weight modification technique to associate such instruction sequences with appropriate domain task behavior. Thus, the strategy of the *Movie Description Network* may be used to transform input sequences of quasi-linguistic tokens into processing element activation vectors which modulate the behavior of a domain task subnetwork. This general approach for "learning by being told" has been tested by application to a simple "discrete mapping" domain.

[3]i.e., advice sequences like those presented in graduate school

4.2 Discrete Mapping

The domain task initially used to explore this strategy for the fabrication of instructable networks was purposely kept very simple. It was an abstract mapping problem of the following form:

> Discrete Mapping - The goal of the system is to map inputs from a finite discrete set to outputs from another finite discrete set. Advice takes the form of temporal sequences of tokens which encode data points in the desired mapping (e.g. "map input A to output B" encoded as the three token sequence "$\Rightarrow$ A B").

In these experiments a vocabulary of three input stimuli, three output responses, and four instruction tokens (including "$\Rightarrow$") were used. All inputs and outputs were specified to the networks using "1-out-of-N" localist codes.

While very simple, this domain task poses interesting problems for the connectionist learning algorithm which is to be employed. The behavior of the system depends entirely on the given instructions. There are virtually no other behavioral regularities which the network may depend upon and discover during training. The system must learn the systematic task of following any arbitrary instruction sequence. If the stream of instruction tokens specify "$\Rightarrow A\ B\ \Rightarrow C\ C\ \Rightarrow B\ C$", then the network must map the domain input corresponding to the "A" token to the "B" output and must map both the "B" and "C" domain input patterns to the "C" output pattern. If, an instant later, the instruction sequence changes to "$\Rightarrow B\ B\ \Rightarrow A\ A$", the system must *immediately* change its behavior to map "A" to "A" and "B" to "B", leaving the mapping from "C" arbitrary (i.e., any output is acceptable). Any sensible instruction sequence (i.e., one that does not require more than one output pattern for each possible input) must be handled by the system. The way in which the discrete mapping problem forces the network to generalize in an exhaustive and systematic manner over the space of instruction sequences makes the problem difficult for the network to learn.

4.3 The Model

The artificial neural network architecture which was initially used for these discrete mapping experiments is shown in Figure 4. The activation level of processing elements in this network was bounded, using a sigmoid, between the values of 0.0 and 1.0. The domain task *input* layer consisted of three elements for the "1-out-of-N" localist encoding of input patterns. The *advice* layer contained four elements for the presentation of similarly encoded instruction tokens — "A" through "C" and also "$\Rightarrow$". The *output* was trained to be a pattern in the same form as the domain inputs. Various sizes for the *plan* layer and for the hidden layers were examined, with a 20 processing element *plan* layer, a 20 element recurrent hidden layer, and a 10 element non-recurrent hidden layer being used for the bulk of the experiments reported here.

The basic idea behind this network design is easy to describe. In short, a simple recurrent network [5] is used to map temporal sequences of instruction tokens into a *plan* vector which is, in turn, provided as input to a feed-forward mapping network. This allows input advice sequences to *immediately* change the discrete mapping performed by the network to be *any* desired mapping. The entire network may be trained by a version of the backpropagation learning law, with error information being provided only for the final output layer.[4] Weight modification learning may be performed on the recurrent connections using backpropagation by "unrolling the network in time" for a single time step [11] — essentially treating the recurrent connections as non-recurrent connections leading to a "context" layer which is made to contain a copy of the hidden layer activation levels from the previous time step [5]. Such a backpropagation learning strategy was used in the experiments which are discussed below, with a fixed learning rate of 0.1 and no momentum. Incremental learning (i.e., "jump every time" or "on-line" learning) was used, causing weights to be modified with every presentation of a supervised error signal.

The network is operated as follows. Before the network receives any input, its recurrent hidden layer has the activation level of all of its processing elements set to an average value (i.e., 0.5). A temporal sequence of input instruction tokens, such as "$\Rightarrow A\ B\ \Rightarrow C\ C\ \Rightarrow B\ C$", is then presented at the *advice* input layer, one token at a time. Activation from these inputs is propagated as far as the *plan* layer, but any further propagation beyond that layer is considered unimportant. When the instruction sequence is complete, the activation levels of the processing elements in the *plan* layer are "frozen" and are used as constant modulating inputs to the mapping subnetwork. Any input patterns then presented at the domain task *input* layer should result in output patterns at the *output* layer which are consistent with the mapping specified by the given instruction sequence. Furthermore, if this process is repeated and a new advice sequence is presented to the network, the mapping behavior of the system

[4]Like the *Movie Description Network*, this architecture was inspired by and is very similar to the *Sentence Gestalt* network [12]. Indeed, it may be argued that the *Sentence Gestalt* network actually performs a "learning by being told" task in a complex question answering domain.

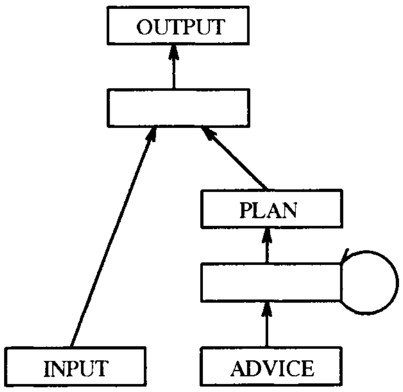


Figure 4 An Instructable Simple Mapping Network Architecture

should *immediately* change to conform to the new instructions. This instructed response is to occur as a direct result of input advice, with *no* additional weight modifications required.

4.4 The Training Procedure

The basic mechanisms of this instructable mapping network are fairly straightforward, but the systematic structure of the discrete mapping problem makes it a difficult problem for this network to solve via error correction learning. Consider that, upon the commencement of initial training, the system will not even be capable of adequately summarizing streams of input instruction tokens into informative plan vectors. The network must simultaneously learn to translate temporal sequences of instructions into useful plan vectors and learn to make use of these plan vectors to modulate mapping behavior. And both of these skills must be acquired using only error information given at the *output* layer during individual mapping trials! It is, therefore, reasonable to expect difficulty during the initial training of this network, and it may be considered potentially profitable to explore non-standard training regimes.

There are many training regimes that may be applied to the network architecture presented here. To expedite the learning process, however, a training regime was selected which maximizes the generation of error feedback to the recurrent plan subnetwork *during* the presentation of an instruction sequence. In general, learning can be expected to be slow if error is available only at the conclusion of complete instruction sequences. Such a strategy would demand that an appropriate set of weight modifications for the proper processing of an entire instruction sequence be computed from a single error signal back-propagated from the *plan layer*. This is a difficult "credit assignment" task which may be partially mitigated by providing error feedback in the middle of advice token sequences. With this in mind, the training regime examined here involved error feedback and weight modification after *every* instruction token presentation according to the following algorithm:

1. Initialize the maximum number of instructions per training session to 1.[5]
2. Randomly choose a number of instructions for this session.
3. Randomly generate the chosen number of instructions.
4. Present the instruction tokens to the network, one at a time. After presentation of each token, freeze the *plan layer* and train the network on each case for which it has seen an instruction. Alternatively, the network may also be trained on cases for which an instruction has been partially presented — for which the full three advice tokens have yet to be seen. (This alternative method which includes partial instructions was used in all of the experiments discussed here.)
5. At the end of each session (collection of consecutive instructions), collect statistics on network performance. After each training period consisting of some fixed number of sessions (e.g., 5000 sessions), compute an average accuracy measure based on the collected statistics. If the system's accuracy is high (above a threshold) then increment the maximum number of instructions per training session.
6. Go to step 2.

Training is terminated when the maximum number of instructions per training session surpasses a fixed threshold. (In the discrete mapping experiments with three possible domain task input patterns, this threshold was three instructions per session.) Note that this training regime is similar to that used for the *Movie Description Network* [2] and to that used for the *Sentence Gestalt* network [12], but this strategy includes a version of the *combined subset training* technique used by the connectionist adders [3]. As much error feedback as possible is provided to the entire network after each advice token is presented, and the

[5] Alternatively, the initial maximum number of instructions may be set at some higher value.

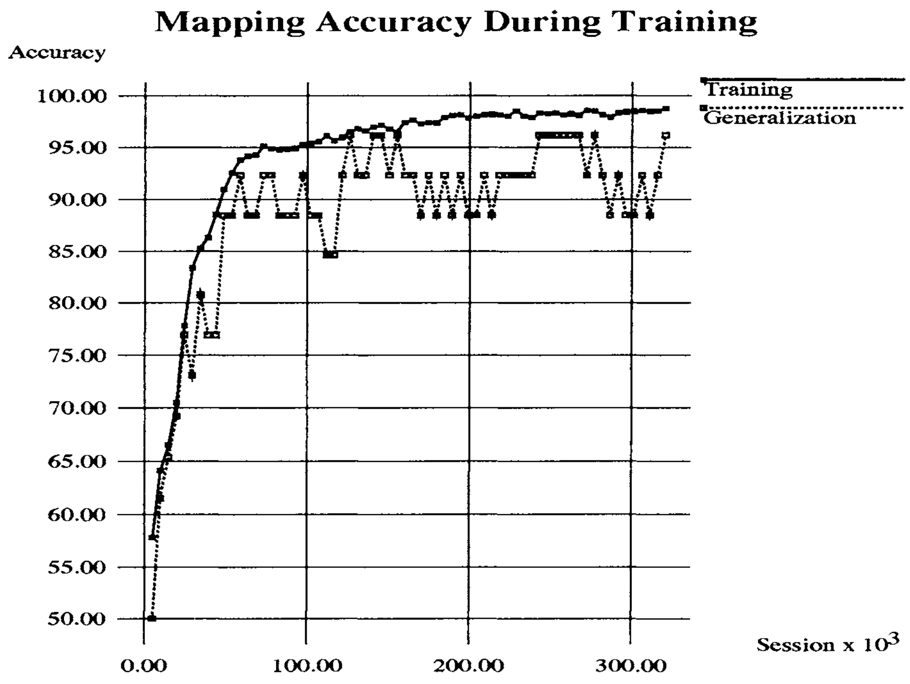


Figure 5 Sample Network Performance

system is given a chance to work up from shorter instruction sequences to longer ones.[6]

4.5 The Results

Many training experiments were conducted using this instructable network architecture and the previously described training procedure, exploring the space of layer sizes and learning parameters. In each experiment, the instructable network was trained on most all those instruction sequences which provided at most one mapping rule for each discrete input stimulus (i.e., consistent instruction sequences). Training was not allowed, however, on elements of

[6]The incremental growth of instruction sequence length actually provided little advantage for the best networks examined here. This is suspected to be a result of the small sizes of the potential input and output sets. The initial maximum number of instructions was set to three for the experimental runs for which learning curves are later provided.

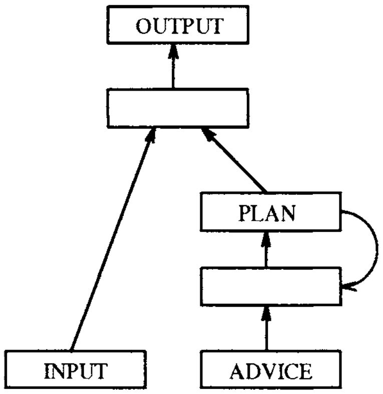


Figure 6 An Alternative Simple Mapping Network Architecture

a generalization testing set which was randomly generated and consisted of 12 instruction sessions (i.e., about 5% of all possible sessions). Instruction sessions were selected at random for presentation to each network. The initial maximum number of instructions per session was set to three.

The performance achieved by the best of these networks is shown in Figure 5. Mapping accuracy was measured after the complete presentation of each instruction session and is presented in this graph as a percentage of the number of input/output mappings which were correctly generated by the network. For the purpose of this measurement, the output element with the highest activation level indicated the network's discrete output response using a "1-out-of-N" localist encoding at the *output* layer. Note that the network eventually achieved on the order of 98% training set accuracy and up to 96% generalization accuracy. Note also that good behavior generally required over 100,000 training sessions. In short, this demonstrated the viability of this strategy, but indicated that more efficient training strategies were required.

In hopes of reducing training time, several architectural modifications were considered. Alternative forms of recurrence were tried, such as variations on that used in Jordan's sequential network [7], as shown in Figure 6. In this architecture, the activation level of the *plan layer* at the previous time step is provided as input to the plan generating subnetwork's hidden layer. This replaces the hidden layer recurrence of the previous simple recurrent network model. Some experiments with networks of this kind showed mild but noticeable improvements in training time, as demonstrated in Figure 7, and pushed generalization accuracy as high as 100%.

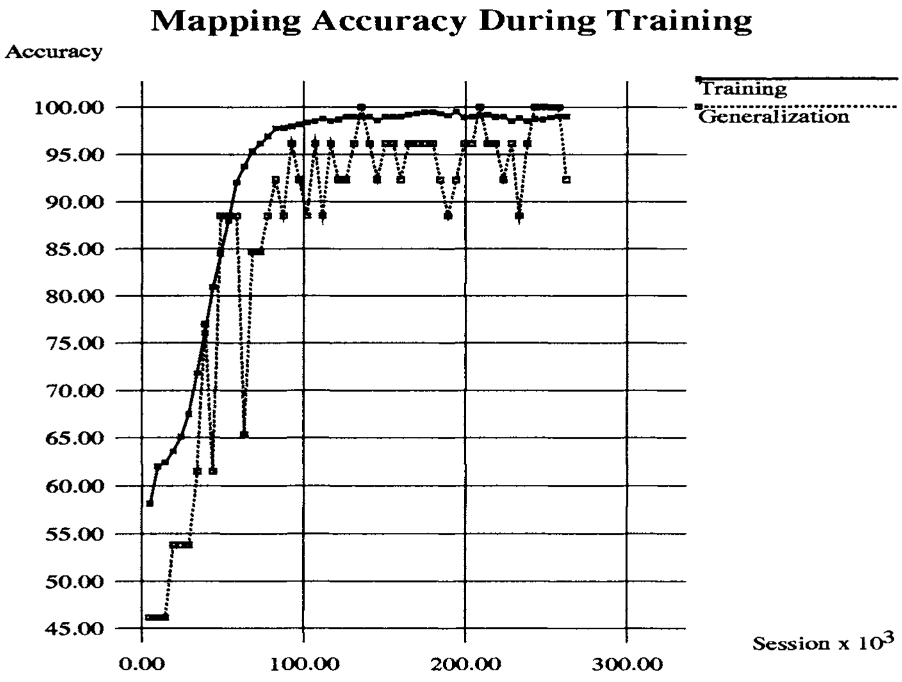


Figure 7 Alternative Network Performance

Another considered design alternative involved the introduction of multiplicative connections into the network, allowing, for example, the plan layer to have a "gating" effect on the mapping subnetwork as shown in Figure 8. Some experiments were conducted in which sigma-pi connections were used to provide such a "gating" effect [11]. Briefly, each possible pair consisting of a domain *input* processing element and a *plan layer* processing element provided the product of their activation levels as input to the non-recurrent hidden layer, weighted by a single trainable weight value. Experiments with such sigma-pi networks (with a reduced learning rate of 0.05) showed slightly improved performance, particularly with regard to generalization accuracy, as shown in Figure 9. Note that this network was able to achieve better than 99% training set accuracy and up to 100% testing set accuracy.

Training time remained relatively high for all of the considered alternative network architectures. There are many options that have yet to be tried,

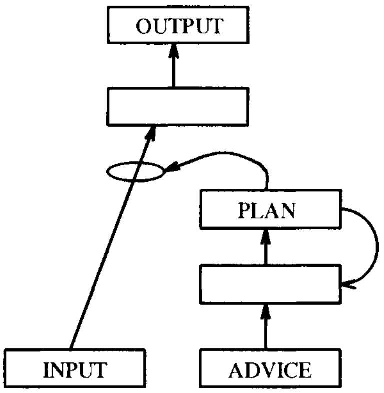


Figure 8 A Network Architecture With Multiplicative Connections

however, including further examination of multiplicative connections, second-order methods, and alternative training regimes.

4.6 THE POSSIBILITIES

In addition to searching for more efficient training techniques for these sorts of instructable connectionist networks, tasks which are more complex than simple discrete mapping should be examined. Specifically, systems which are to perform tasks in some spatiotemporal domain should be generated. As in the discrete mapping task, instruction sequences should be used to modulate the behavior of the system, but the behavior in question should now have a temporal dimension.

Experiments are currently being conducted which involve the modulation of the systematic behavior of an arithmetic network, similar to the adder networks, by a sequence of quasi-linguistic instructions. The architecture which is being used for these experiments is shown in Figure 10. The system is to perform arithmetic operations on two arbitrarily sized numbers in much the same way that the adder network performed addition, but the exact "program" that the system is to follow is to be specified as an input sequence of instruction tokens. Different instruction sequences may specify different orderings for sets of standard actions (e.g., announcing the carry before or after recording the digit sum) or may specify completely different arithmetic operations (e.g., subtraction rather than addition). As might be expected, training is a slow process

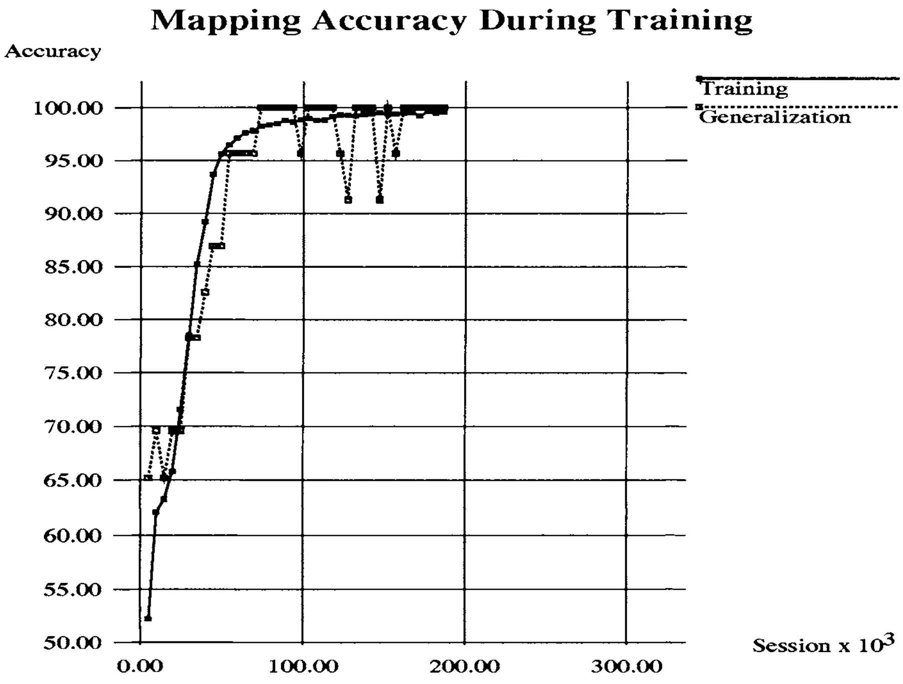


Figure 9 Sigma-Pi Network Performance

for this network. While early experiments are successfully generating systems capable of enacting "programs" involving several variations of addition and subtraction, summary results are not yet available.

5 SUMMARY

Some small initial steps have been taken towards connectionist systems which exhibit the systematic flexibility of advice following without abandoning the power of experiential inductive generalization. Such systems, which are capable of both learning from examples and "learning by being told", could potentially help cognitive scientists explain "high level" reasoning processes and provide insight into how such processes may emerge from more simple associational mechanisms. A basic strategy for instructable connectionist systems has been outlined, including a focus on recurrence for systematic-

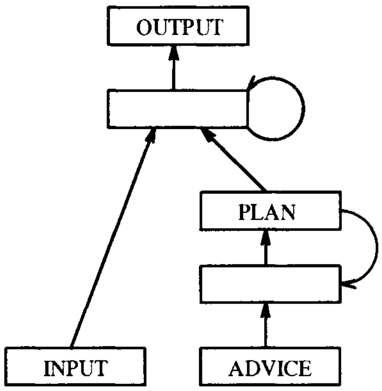


Figure 10 An Instructable Arithmetic Network Architecture

ity, learned temporal patterns for linguistic interaction, and activation state modification for fast behavioral change.

Recurrent networks are necessary to generate complex systematic behavior in a connectionist framework. Any task which requires the iterative memorization and retrieval of internal representations will require some form of recurrence. Such manipulation of internal state information is common for systematically structured domains. Fortunately, there is currently much research being conducted involving the dynamic properties of recurrent connectionist systems and involving the training of such systems. The explorations that have been conducted here have shown, however, that much can be accomplished even with a few relatively simple network architectures. These networks, while conceptually simple, are capable of learning complex time-varying behaviors by leveraging the power of distributed representations generated by the well studied backpropagation learning algorithm. Contrary to some critics of connectionism, systems based on these distributed architectures are not only capable of *performing* systematic behaviors, but they are also capable of *learning* such behaviors.

Any instructable connectionist system must be able to receive and process quasi-linguistic statements. Strategies involving the *compilation* of linguistic instructions into network weights are generally too restrictive to be of use in cases where instruction is to be more than a one shot occurrence. By encoding linguistic statements as temporal activation patterns, and by allowing domain task specific internal distributed representations of these patterns to form as a result of inductive learning, connectionist systems are granted the power to interact linguistically with the world and to have language interact appropriately with their actions. As demonstrated by the simple instructable networks which were discussed here, linguistic input may be made to modulate a systematic behavior. It is also possible that recurrent networks performing a systematic task may potentially be trained to *explain* their systematic behavior in a quasi-linguistic fashion, thereby revealing their inner workings. Encoding linguistic statements as time-varying distributed patterns of activation opens the door to many opportunities.

If an instructable connectionist system is to modify its behavior *immediately* in the face of advice, only two general design strategies are possible. Either some sort of fast weight modification mechanism must be installed, or behavior must be modulated by changes in the activation state of processing elements. The strategy which is described here leaves the process of weight modification in the capable hands of inductive learning algorithms and focuses instead on encoding the receipt of instruction as motion in an inductively learned distributed activation space. Metaphorically, "learning by being told" involves an instruction sequence input pattern *pushing* a network into a new region of activation space — a region corresponding to the desired behavior.[7] Restricting network activity to this region of activation space effectively restricts network behavior, as well. A connectionist system which has successfully learned the regional topology of this activation space will be capable of responding to instruction as rapidly as activation levels can change.

While this proposed strategy for connectionist advice taking shows much promise, it is certainly not without flaw. The main thing that is missing from this account is, as outlined in the introduction, memory. In order to finesse this problem, in this work we simply "froze" the network activations at the *plan layer*. Adding a memory to this model should be seen as the first goal of future work. It may be trivial to add an appropriate attractor structure to the network, but it is more likely that such attempts will contain unforeseen difficulties. Also, critics of a simple attractor model of memory may attack such attempts by referring to related psychological phenomena. For example, one can remember, albeit briefly, nonsense words like "frobitz", for which there is no clear reason to expect an attractor bowl in our lexicon. There are several possible responses to such criticism. First, "frobitz" is much more similar to valid English words than "mxytlpx", which has the same number of letters but is much less memorable. This supports an attractor based memory model in which general phonological features help shape attractor basins. Second, it may be necessary to add special purpose memory devices. One memory model

[7] Thanks are due to Paul Churchland for this observation [1].

that makes fast associations using an activation-based approach is Metcalfe's CHARM [9] model. There is plentiful evidence to indicate that the human memory system involves many specialized components. Can less be expected of a connectionist cognitive model?

Another obvious problem with this approach to "learning by being told" is the difficulty with which networks of this kind learn the needed language interpretation skills. More efficient training techniques are needed for recurrent networks performing systematically structured tasks if this strategy is to be applied to reasonably sized networks. Specifically, variations on *combined subset training* should be examined. Also, learning to take advice should be factored out from the task of learning to operate in a given domain. There are many things that could be learned in advance about the regularities of a domain before advice is proffered. This should make the process of learning to take the advice easier and correspondingly faster.

As these issues are resolved by further research, perhaps cognitive modelers will not sense a need to retreat to hybrid symbolic/connectionist systems to solve their knowledge acquisition and manipulation problems. In the end, connectionism might be found to provide it all.

ACKNOWLEDGEMENTS

The ideas described in this document were found in the intersection of the work of several insightful researchers. The generation and analysis of the adder networks which were described here was, for the most part, performed by Fu-Sheng Tsung [3]. The *Movie Description Network* was the result of work by Brian Bartell [2]. Thanks are also due to Paul Churchland for his insights concerning the relationship between instruction sequences and regions of network activation space.

REFERENCES

[1] Paul M. Churchland. *Talk presented at the Konnektionismus In Artificial Intelligence Und Kognitionsforschung Conference.* September 1990. Salzburg, Austria.

[2] Garrison W. Cottrell, Brian Bartell, and Christopher Haupt. Grounding meaning in perception. In H. Marburger, editor, *Proceedings of the 14th German Workshop on Artificial Intelligence*, pages 307–321, Berlin, 1990. Springer Verlag.

[3] Garrison W. Cottrell and Fu-Sheng Tsung. Learning simple arithmetic procedures. *Connection Science*, 5(1):37–58, 1993.

[4] Lawrence Davis. Mapping classifier systems into neural networks. In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 49–56, San Mateo, 1989. Morgan Kaufmann.

[5] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[6] Jerry A. Fodor and Zenon W. Pylyshyn. Connectionism and cognitive architecture: A critical analysis. In Steven Pinker and Jacques Mehler, editors, *Connections and Symbols*, pages 3–72. The MIT Press, Cambridge, 1988.

[7] Michael I. Jordan. Serial order: A parallel distributed processing approach. Technical report, Institute for Cognitive Science, UCSD, 1986.

[8] James L. McClelland, David E. Rumelhart, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2. The MIT Press, Cambridge, 1986.

[9] Janet Metcalfe. Recognition failure and the composite memory trace in CHARM. *Psychological Review*, 98:529–553, 1991.

[10] Ryszard S. Michalski and George Tecuci, editors. *Machine Learning: A Multistrategy Approach*, volume 4. Morgan Kaufmann, San Mateo, 1993.

[11] David E. Rumelhart, James L. McClelland, and the PDP Research Group. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. The MIT Press, Cambridge, 1986.

[12] Mark F. St. John and James L. McClelland. Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, 46(1-2):217–257, 1990.

[13] Volker Tresp, Juergen Hollatz, and Subutai Ahmad. Network structuring and training using rule-based knowledge. In S. J. Hanson, J. D. Cowan, and C. L. Giles, editors, *Advances in Neural Information Processing Systems 5*, San Mateo, 1993. Morgan Kaufmann.

7

An Internal Report for Connectionists

NOEL E. SHARKEY AND STUART A. JACKSON

Computer Science Department
University of Sheffield
S1 4DP, Sheffield, UK

1 INTRODUCTION

Although there is a considerable diversity of representational formalisms in the Connectionist literature, most of the excitement about representation has been concerned with the idea of distributed representations. Dissatisfied with the Symbolic tradition, and in search of the new, many cognitive theorists began to infiltrate connectionism in search of a new theory of mind. Like the Classicists, these theorists required that a constructed, analytic theory of mind postulate complex structured representations; it is only by having *structural* similarities among representations that we can account for the systematic nature of human thought. The Classical view is that the systematic relations between representations are necessarily based on the composition of similar syntactic elements. Likewise, some of the representational types found in the connectionist literature satisfy this requirement, only by virtue of the fact that they are similar to Classically conceived symbolic representations. For example, the structure of complex expression may be maintained in vector frames consisting of explicit tokens or complex expressions that are essentially passed around a net [18]. Only distributed representations offered the promise of a novel representational scheme that could underpin a connectionist theory of cognition; a scheme that relied upon the assumption that structural similarity relations can be captured in a connectionist net by systematic vectorial distances between the distributed representations created.

This very idea, that spatial (Euclidean) similarity relations could capture systematic structural similarities, serves to separate the Connectionist and Classical theories. For the Classical theorist, the necessary and sufficient structural similarity relations for a representational theory of mind are *syntactic*, whereas for the Connectionist, they are *spatial* (ie. given in terms of similarity of location in a Euclidean space). Indeed, that representations can stand in non-syntactic similarity relations is *crucial* to the emergence of a Connectionist theory of mind. The current, ongoing debate between the Classical and the Connectionist schools, regarding their different conceptions of the components of a theory of mind, has turned *precisely* upon this issue, namely, upon the nature of the requisite structural similarity relations that hold between representations. A great deal of effort has gone into devising an explanatory framework that makes use of spatial, as opposed to syntactic, structure, as witnessed by the explosion of descriptive terms. Thus, *fully distributed representations*, *superpositional storage*, *holistic computation*, *non-concatenative compositionality*, and indeed, *spatial structure* itself, are all new terms either invented for the purpose, or ported into Connectionism in order to distinguish and explicate a non-symbolic theory of mind.

In this chapter, we focus, not upon the terms of the debate between the Classical and the Connectionist schools, but on the underpinnings of the emerging connectionist theory of cognition. We challenge a key assumption of the proposed connectionist representational scheme; that the precise distances between vectors of distributed representations, learned by a connectionist net, reflect systematic semantic and/or structural similarity relations. On this understanding, even a *minute* difference in the distance between one representation vector and any of the others will change the 'meaning' of that representation (cf. [11], [3]). The structure of our argument will proceed as follows: In Section 2, we examine the origins of Connectionist representations, from their humble beginnings to their most modern sophistication. Along the way, we shall see the emergence of the key 'spatial systematicity' assumption (ie. the assumption that systematicity has a spatial explanation). In Section 3, we present a detailed argument using constructive methods, drawing upon a simple decision space technique (cf. [20]), which demonstrates that this assumption may not be warranted except under special circumstances. We will show here that the computational role of a, so called, distributed representation may be separated from its specific neighbourhood distance relations (where a neighbourhood is a group of 'close' points defined by some Euclidean distance metric). This analysis will then be used to support an argument in favour of 'whole net' representation. In Section 4 of the paper, we consider the implications of our findings, and consider a possible re-orientation toward the notion of functional representations in which the input is represented by the whole net, weights as well as units, in terms of its computational role.

2 The Origins of Connectionist Representation

The term *distributed representation* grew out of the idea of how concepts or representations, let us call them entities, from a descriptive language could be recoded into binary vectors (bit vectors) such that, (a) an entity that is described by a single term in the descriptive language is represented by more than one element in the connectionist representation ([13]). For example, if the lexical item 'ball' was a term in the descriptive language, then the distributed representation would have 'bits' to represent, say, the *features*, or *microfeatures* as they are often called, of 'ball' including 'round' and 'bouncy'; (b) each of the elements in the distributed representation must be involved in representing more than one entity described by a single term in the descriptive language. For example, the (micro)feature 'round' for the item 'ball' may also be used as part of the distributed representation for other items such as 'Edam cheese'.

These distributed binary vectors can be thought of, in a very restricted sense, as representations with content in that each of the elements of a binary vector stands for a content element of the entity being represented. However, despite the use of the term *microfeature* instead of the Classical term *feature*, featural representation had already been used extensively in the Symbolic tradition (e.g. phonological features, and semantic features) as had the notion of distance in feature space (e.g. semantic distance, [4]). There was not a lot of new action for the representationalists. The novelty of connectionism was in the way in which higher order relationships could be extracted from the featural representations and how these representations could be put to work in a variety of tasks. Most types of Connectionist representation were similar to Classically conceived symbolic representations. Some of these consisted of explicit tokens or complex expressions that are essentially passed around a net, while others maintain the structure of complex expressions in frame vectors. This prompted a severe and scathing attack on the idea of a new connectionist theory of cognition from [9]. One of their main arguments was that connectionist representations are either hopelessly inadequate for the purpose of developing a cognitive architecture, or they are merely, at best, implementations of Classical representations.

The Fodor and Pylyshyn criticism also had a positive side. Following its publication, all hands went to the pumps, and before long very many replies began to emerge from the connectionist community (look at any collection of cognitive connectionist papers between 1988 and 1992). The representationalist, in search of novelty, began to focus on the idea of *fully* distributed or superpositional representations. These are vectors of continuously valued vectors such as the 'hidden unit' vectors used by multilayer learning techniques such as backpropagation ([17]) and its variants (e.g. [21], [54]), or the type of superposed unit activations developed using Tensor Products ([15]). Such representations are called *superpositional* because the representings of two distinct items are superposed if they occupy the same portion of the unit resources available for representing (a clear treatment of superposition is given in [12]). In order to define superposition more formally, we might say, following [1], that **R** is a representation of individual content items c_1, c_2, ... c_n if, (1) **R** represents each c_i; (2) **R** is an amalgam of representations r_1, r_2, ... r_n of content items c_1, c_2, ... c_n respectively, and (3) r_1, r_2, ... r_n each uses the same unit resources in representing a given content c_i.

The discovery of such novel representations was cause for much excitement in the cognitive science community because they, at last, had an alternative to the symbolic representations of Classical AI, one that also offered a non-symbolic alternative to the simple input/output contingencies of behaviourism. Nonetheless, there was still considerable disquiet in the Classical camp. The main bone of contention concerning superpositional representations was that, according to some, they lacked compositional structure. In a critique of uniquely connectionist representations, [8] argue that in order to support structure sensitive operations, complex representations must, literally, contain explicit tokens of the original constituent parts of the complex expression:

> '...when a complex Classical symbol is tokened, its constituents are tokened. When a tensor product or superposition vector is tokened, its components are not (except *per accidens*). The implication of this difference, from the point of view of the theory of mental processes, is that whereas the Classical constituents of a complex symbol are, *ipso facto*, available to contribute to the causal consequences of its tokenings - in particular, they are available to provide domains for mental processes - the components of tensor product and superpositions vectors can have no causal status as such.'
> (Fodor & McLaughlin, 1990, p. 198.)

These criticisms forced the Connectionist community to examine more closely what [20] have called the three horns of the representational trilemma facing cognitive science: That is, whether distributed representations are (or could be) *compositional*, whether the kinds of putative mental processes that distributed representations enter into could be *systematic* and (to a lesser extent for the purposes of this chapter) whether distributed representations could be *grounded.* However, we must tread warily when we label a new entity in science lest we smuggle inappropriate terms and theoretical attributes behind the guise of the label. Compositionality and systematicity are both terms borrowed from the study of artificial and natural languages in model theoretic semantics and linguistics, and applied to the study of the putative Language of Thought (cf. [7]) in cognitive science. *Linguistic* capacities are said to be systematic, in that the ability to understand some sentences, for example, 'Jerry likes Connectionists', is intrinsically connected to the ability to understand certain others, for example, 'Connectionists like Jerry'. Moreover, this capacity is identified with the feature of all languages that its constructs (viz. sentences) have syntactic and semantic structure. The argument translates directly to the LoT. Thus, *cognitive* capacities are said to be systematic, in that the ability to *think the thought* that, for example, Jerry likes Connectionists is intrinsically connected to the ability to think certain others, for example, the thought that Connectionists like Jerry. Moreover, this capacity is identified with the feature of the putative LoT, that thoughts have syntactic and semantic structure. Thus, just as there are structural similarities between the sentence 'Jerry likes Connectionists' and the sentence 'Connectionists like Jerry', so too there must be structural similarities between the mental representation that Jerry likes Connectionists and the mental representation that Connectionists like Jerry. The conclusion that is drawn from these examples is that mental representations, just like the corresponding sentences, must be constituted of the same parts in a systematic compositional manner.

So how does all of this bear on distributed representations? Well, the charge is made (cf. [9]) that distributed representations are not compositional and thereby they are not able to support systematic cognitive capacities. To be more specific, the charge is made that distributed representations are not candidate mental representations because they do not have a combinatorial syntax and semantics. That is, distributed representations make no use of the distinction between structurally molecular and structurally atomic representations, they do not have tokens of constituents as literal parts and the content of a distributed representation is not a function of the content of its literal parts together with its constituent structure.

Since the publication of their article however, Fodor & Pylyshyn have been shown to be a little bit premature in their conclusions. Specifically, what has emerged from the Connectionist literature is that distributed representations *can be compositional* without themselves being Classical. To recall, the primary requirement of a representational theory of mind is that *representations have internal structure*, or that they have a significant *constituent structure.* The Classical theorist argues that such constituent structure is *syntactic*, resulting from a *concatenative* method of combining constituents. However, the insight that van Gelder ([11]) disseminated was that Connectionist representations *can have* internal structure, without that structure being syntactic. What legislates for this assertion is the realization that concatenation is not the only means available to the computational theorist for constructing complex compounds. Thus, instead of employing a concatenative method of combining constituents to form compounds, which then have a resultant *syntactic structure* (as the Classical theory of mind asserts), the Connectionist theorist employs a non-concatenative method of combining constituents to form compounds, which then have a resultant *spatial structure*. Thus, instead of the syntactic similarity relations that hold between Classical symbols, the Connectionist theorist appeals to spatial similarity relations, similarities usefully understood as similarities of *location* in decision space. The proposal is that it is the specific distances between the representations that provides Connectionism with a structured compositional scheme.

Much of the force of the Classical attack on Connectionism is deflected when one realizes that it is possible to entertain the kind of *qualitatively different* spatial structure similarity relations that non-concatenative, as opposed to concatenative, methods of combination provide for complex Connectionist representations. When discussing the internal structure of vectors of activation, it is important to be aware that, on current understanding, a compound Connectionist representation is not regarded as having tokens of constituents literally present within it (as the compound Classical representation does). Rather, representings of constituents are superposed within the compound representation itself, as we have already discussed. [1]

It is these insights, of the existence of non-concatenative compositionality and spatial structure similarity relations, which inform a fundamental assumption of the Connectionist theory of mind. That is:

> . . . the *particular* internal makeup of a *given* representation determines how it stands to other representations. (Van Gelder, 1989)

[1] The identity of constituents in fully distributed representations are destroyed in the process of composition. Whilst not literally tokened, the constituents are still said to be causally effacacious in computation. Indeed, Sharkey ([19]) provides a method for extracting the representations of constituents out of holistic representations of the inputs. The implications of the current paper for this method of extraction are not addressed here.

For van Gelder, it is the distance relationships between the representations which is of paramount importance. Later in the same paper, he states and re-stresses the point that:

> The position of a representation in the space has a semantic significance; vary its position in that space, and you automatically vary the interpretation that is appropriate for it ... Representations with similar sets of constituency relations end up as 'neighbouring' points in the space. The systematic nature of these spatial relations can be discerned using sophisticated techniques for the comparison of large groups of vectors such as cluster analysis. (Van Gelder, 1989)

In this chapter, we intend to challenge this assumption of 'spatial systematicity', by showing that neighbourhood analyses, such as hierarchical clustering techniques, only correlate with the computational properties of a net, they do not show the causal consequences of distance on the computation. This challenge could have serious consequences for the emerging Connectionist theory of mind, because as van Gelder says:

> . . . all this counts for nothing unless these [distance] relationships *matter*...unless the location of a given representation in the space is such as to be of some kind of systematic semantic and computational significance. (van Gelder, 1989)

Our purpose here is not to burn what one might call a Spatial Strawman. Instead, we intend to show how an alternative form of systematicity might arise in multilayer nets as a result of the interaction between the input weights and the output weights. This is a functional form of systematicity in that it relates the inputs, not to each other in some representational space, but to their function on the output. We proceed with our argument by showing how the computational role of input patterns can be separated from their similarity in representation space.

3 Representation and Decision Space

In order to show some of the technical problems associated with the notion of spatial systematicity, we focus here only on nets with two weight matrices and three layers of units (input, hidden, and output) trained using the backpropagation learning rule (but the analyses applies equally well to other types of nets). In order to avoid importing the theoretical spectacles of representationalism into our arguments, we use the neutral term *address* rather than *hidden unit representation, fully distributed representation*, or *superpositional representation* to refer to a vector of hidden unit activations, $\mathbf{h} = [h_1, h_2, ..., h_n]$. The elements of the address vector are each coordinates for a point in an n-dimensional decision space (described below). This is the addressed *location* of an input vector. For example, the address for the input vector [1,1], in Figure 2, is [0.05, 0.91]. Thus, for a net with two matrices of weights, there are two processing steps. The first step is to allocate an address for a location in decision space to the input vector. For the second step, the outputs are determined by the relation of the addressed location to the hidden-to-output weights.

It is possible to visualise this relation using decision space analyses which provide graphical representations of network computation. Such analyses have been used extensively for networks with a single matrix of weights between the inputs and outputs. These make a good starting place to explain the technique. Before discussing multilayer nets, we first examine some simple binary nets like those developed by McCulloch and Pitts (1943) for the basic functions: P AND Q, P OR Q, and NOT P AND Q. The nets are shown in Figures 1(a)-(c). The output function is the Heaviside or threshold function, where an output $o = 1$, if $\mathbf{w}.\ \mathbf{v} > \theta$ (where θ is a value between 0 and 1, $\mathbf{w}$ is a weight vector and $\mathbf{v}$ is an input vector). The input space for the nets shown in Figure 1 is two dimensional and can thus be described as a square with the binary inputs arranged on its vertices. Each vertex can be thought of as a vector from the origin. To see what the net computes, the weights to the output unit may also be plotted as a vector from the origin. A decision line, perpendicular to the weight vector, can then be drawn through the input space by solving the equation, $xw_1 + yw_2 = \theta$, for x and y, where w_1 and w_2 are the two weights, and x and y are coordinates of the decision boundary. The line divides the square into two *decision regions* that show which of the inputs will produce a +1 as output and which will produce a zero.

Figures 1(d)-(e) show the decision regions for the corresponding nets in Figures 1(a)-(c). The diagrams entirely determine the computation of the nets. For example, if continuous rather than binary values were used as input to the nets, the decision line would show the regions of generalisation of the net i.e. which continuous valued inputs would map onto a +1 output and which would map onto a zero output. For nets with multiple output there would be one decision line for each output unit.

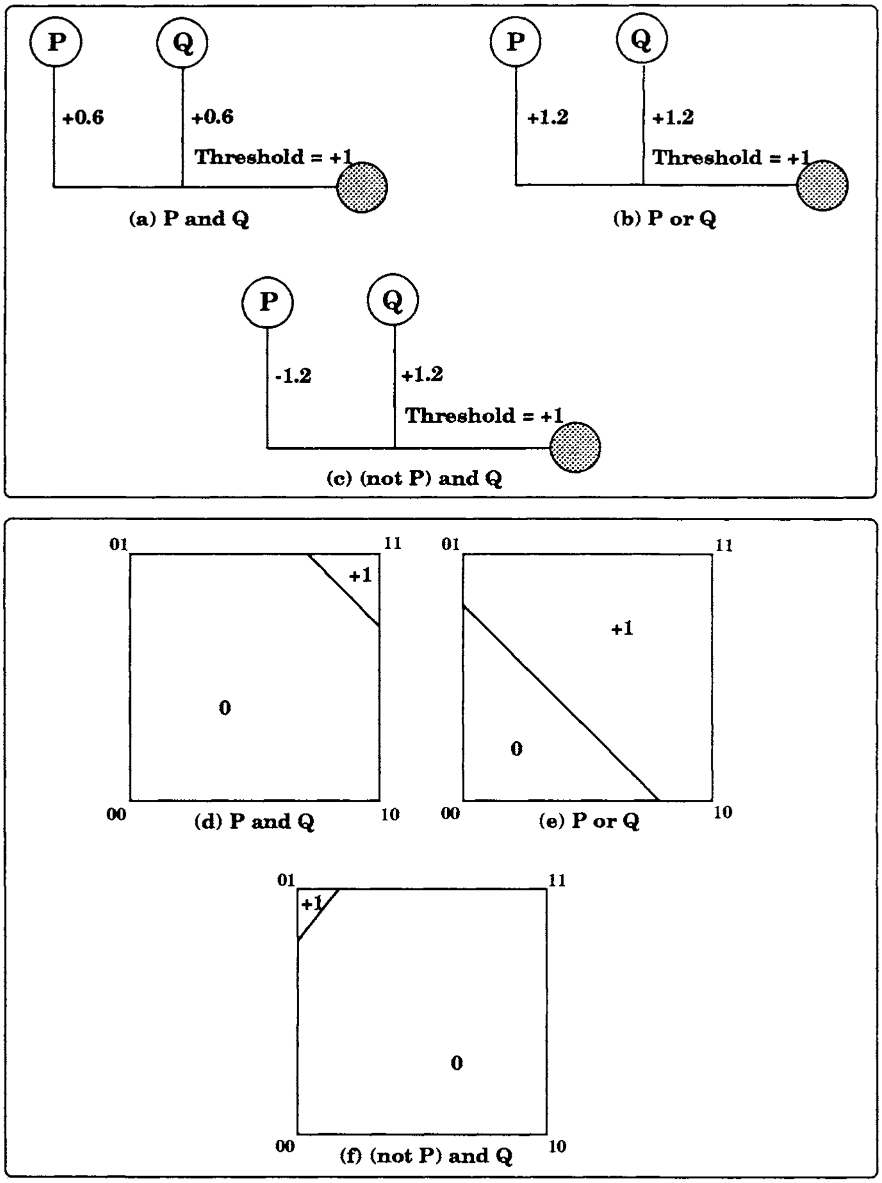


Figure 1 Top: Three 'McCulloch-Pitts' binary nets for the three functions: P AND Q, P OR Q, (NOT P) AND Q, and bottom, the three decision regions for the corresponding functions.

In a network with a single matrix of weights the input points are all fixed in advance and so training is limited to moving the decision lines around the input space until all of the points are captured in the required regions. Having such fixed input points restricts the functions that such a net can compute to those in which the pattern classes are linearly separable. This is a severe restriction since the ratio of linearly separable to linearly dependent pattern classes rapidly approaches zero as the dimensionality of the input space increases. For example, it is not possible to move a line around the 2D input space to separate the regions appropriately for XOR function ($11 \rightarrow 0, 00 \rightarrow 0, 10 \rightarrow 1, 01 \rightarrow 1$). One solution is to translate the input points into a new space such that they can be separated appropriately by the decision line. This is the solution strategy used by backpropagation learning in multilayer nets.

Unlike training a net with a single weight matrix, a feedforward net with two matrices is not restricted to moving decision boundaries around fixed input points. It can move the input points as well, or at least their addressed locations. Initially each weight in the network matrices is set to a random value (usually between -0.1 and 0.1). Each input vector is passed through the two operations: allocate an address and generate an output. The *actual* output is compared with the *required* output and, if it is incorrect, an error signal is passed to the output weights to move the decision boundaries towards addressed locations that should be in those regions and away from locations that should not. An error signal is then passed to the input weights so that the addresses are altered to move the locations closer to appropriate regions and away from inappropriate regions. A simple macroscopic description of backpropagation learning is that the addressed locations are attracted by their appropriate decision regions and repelled by their in appropriate regions. Likewise, the decision boundaries are attracted and repelled by the addressed locations. During learning the decision boundaries and the addressed locations are juggled until the input classes are separated appropriately by the decision boundaries and the points are outside of the sphere of influence of repellor regions.

An important point is that as soon as the net has separated the classes, as described, the process terminates. There is no fine tuning of the individual distances between the addressed locations. The main determinants of where an addressed location will end up in decision space (apart from rate parameters) are: (i) the initial addresses of the inputs; (ii) The initial positions of the decision boundaries; (iii) similarity relations in input space; and (iv) the required similarity relations in the output space. By way of example, Figure 2 shows a decision space for a fully connected feedforward net that was trained on the XOR task using backpropagation learning. Note how the input patterns have been allocated addresses in the decision space that allow them to be separated by a line. This shows clearly that a net with two weight layers does not necessarily implement a nearest neighbour classifier. Two addressed locations with the same computational roles may be relatively distant as shown for the input vectors [0,1] and [1,0] in Figure 2. The point is that being in the same neigbourhood often correlates with being in the same decision region, but the two are separable. It is not the distances between the addressed locations that determines the computation, rather what the net will compute for a given input is entirely determined by its addressed location in relation to the decision boundaries.

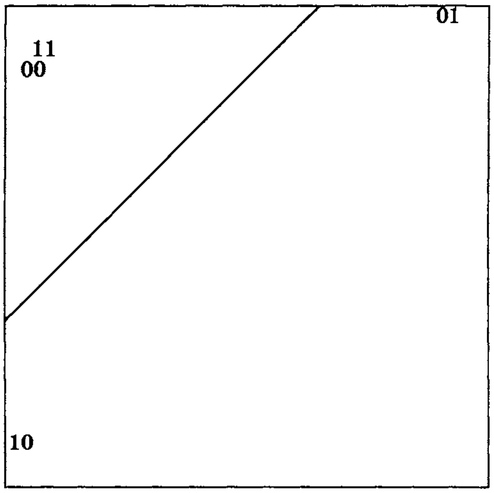


Figure 2 The decision regions for a net trained on the XOR function.

This latter point will be demonstrated by working through some examples. An address space is illustrated in Figure 3 with the coordinates of twelve input patterns shown as numbered locations. [2] These locations are not, as yet, labelled according to their input or class. Since the labels do not have a causal role in the computation, their ommission enables us to look at the data neutrally. *Post hoc* explanations of why, for example, 'tomato' was clustered with 'cucumber' instead of 'apple' can be very distracting.

[2]Backpropagation tends to push the hidden unit vectors towards the axes of the space. It is also possible to find instances of backpropagation learning in which some points are located in the middle of the space, for example, where they are novel inputs patterns and they are orthogonal (or nearly orthogonal) to the trained patterns as illustrated in the examples.

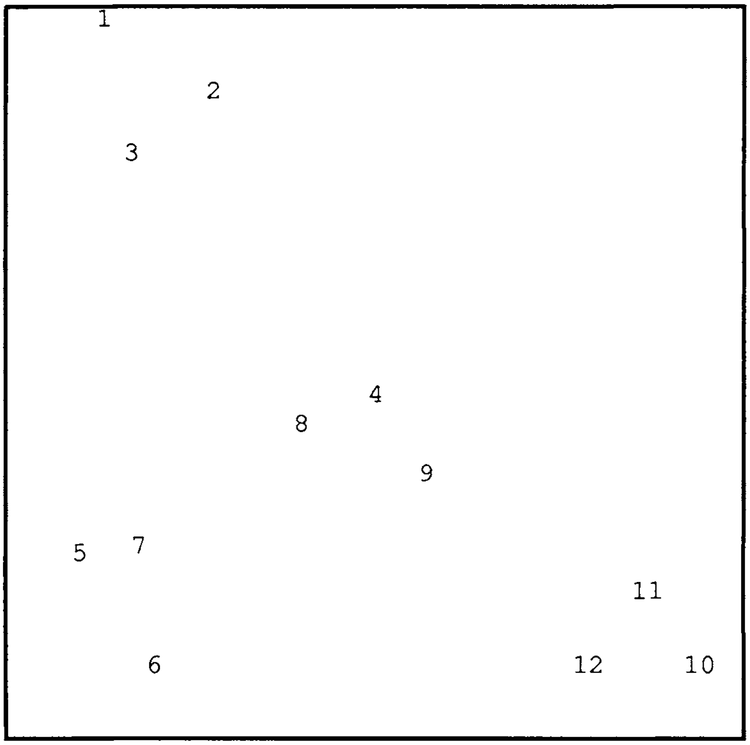


Figure 3 An address space with numbered locations.

A visual inspection of Figure 3 reveals four clusters of locations in the space: (i) 1,2,3; (ii) 4,8,9; (iii) 5,6,7; and (iv) 10,11,12. This is supported by a Hierarchical Cluster Analysis of the squared Euclidean distance between the points. A dendogram of the analysis, shown in Figure 4, only differs from the visual inspection in linking two of the clusters, (ii) and (iv), in a central superordinate cluster. We could speculate that the clusters represented types, for example, *animal*, *vegetable* and *mineral*. The central (superordinate) cluster in the dendogram could be showing that the two clusters are subordinate types such as *mammal* and *non-mammal*. Alternatively, the clusters could represent *agents*, *verbs*, and *actions*, with the verbs subdivided into *transitives* and *intransitives*.

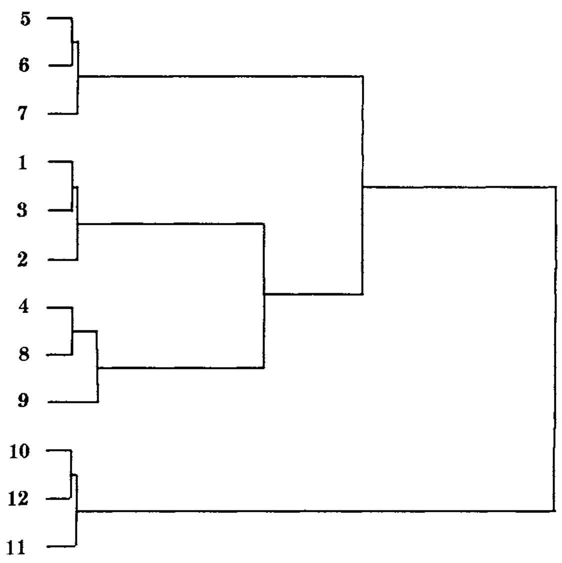


Figure 4 A dendogram of the hierarchical cluster analysis of the data points shown in Figure 3.

We can now use our speculations to show how a neighbourhood analysis, such as cluster analysis, can yield quite different results from a computational analysis. To complete the picture, the weights to each of the output units are used to draw decision boundaries through the input space. The resulting decision space diagram is shown in Figure 5.

Figure 5 tells us that there are three output units (because there are 3 decision lines) and that they divide the space into a number of positive regions containing the addresses for points 1-4, 5-8, 9-12. This differs considerably from the neighbourhood analysis. Since the decision space determines the computational type of each of the input tokens, it shows that the interpretation of the neighbourhood analysis was incorrect. Although the results of the cluster analysis were correlated, to some extent, with the decision space, they did not provide a reliable measure of the computational roles of the inputs. [3]

[3] In this chapter we describe only the common binary output nets typical in cognitive modelling. With continuously valued output units we may think of each output unit as implementing a series of boundaries each of which reflects a particular value on the output. With such a net the outputs are fixed by the intersection of the boundaries. For example, this is how the

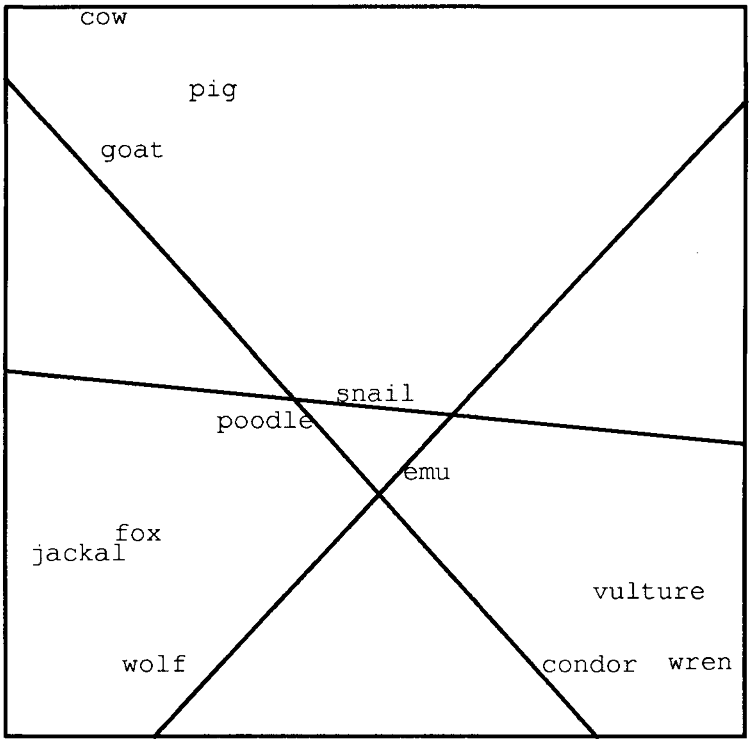


Figure 5 A decision space for the numbered points in Figure 4.

When cluster analyses are shown in the literature, the labels associated with the inputs are usually shown. These labels can often provide information about the computational role of the inputs and are, in a sense, often used in place of a proper computational analysis, e.g. we might know that *tiger* should cluster with *lion* rather than with *comb*. In Figure 5, labels are associated with each of the address points and decision lines from Figure 3. We can see now that the space is divided up into *canines*, *herbivores*, and *birds*. Each of the

non-terminals of a RAAM net ([54]; [2]) work. However, there is still a legitimate separation between specific Euclidean distance relationships in hidden unit space and the computational roles of the inputs. Even then, in terms of the overall functionality of a RAAM, the correlation between distance and function arises as a result of similarities between the input terminals rather than between structurally similar input expressions.

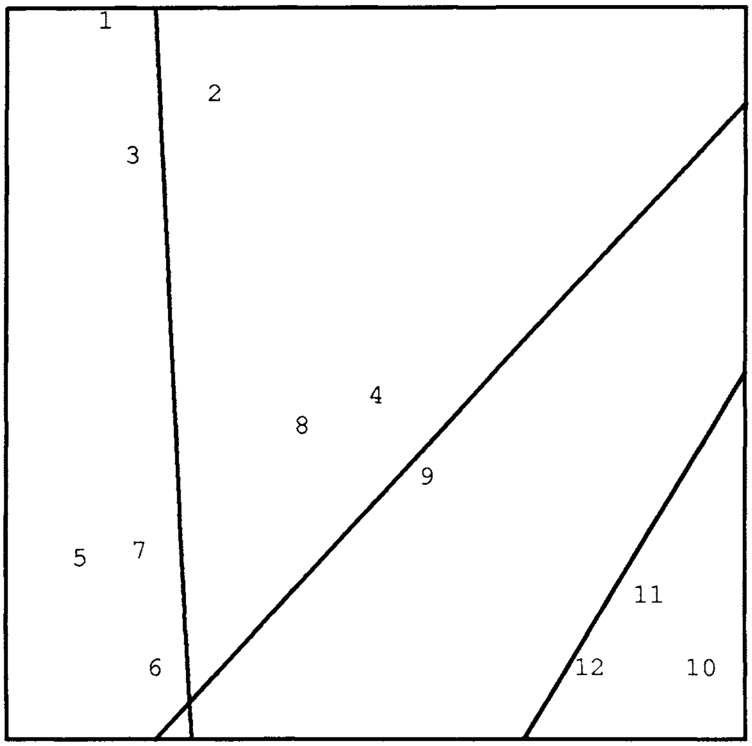


Figure 6 An alternative decision space for the numbered points in Figure 4.

addressed locations falls into one of these regions and thus we can say that each of the input patterns falls into one of those types. Now, this analysis highlights a common misconception in the literature (e.g. [18]) that the addresses are representations of the input that contain type information. This is true only in that the coordinates do locate the addresses within particular regions. However, it should be very clear at this point, and must be stressed, that types are not implemented by their specific distance relationships within neighbourhoods.

The force of this point can be seen by drawing new decision lines through the decision space as shown in Figure 6. Three new decision lines have been drawn through the space so that the type structure is now different than it

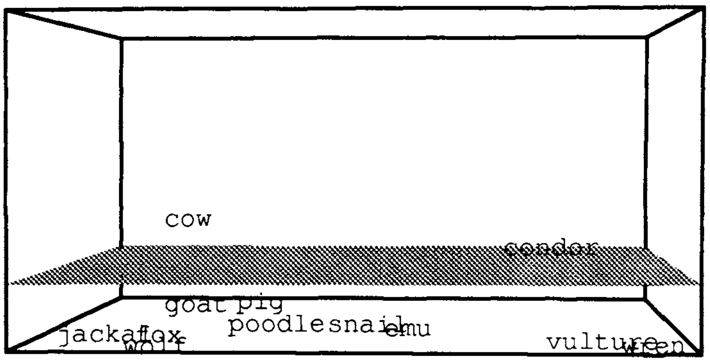


Figure 7 View of the decision space of Figure 6, viewed in 3D with the addition of the z axis.

was in Figure 5. Now locations 1,3,5,6,7 share the same computational role, distinct from the computational role shared by locations 2,4,8 and distinct from the computational role shared by locations 10,11,12 (Location 9 has its own, unique computational role). In other words, the type membership of a location can change even though its vector of hidden unit activations remains constant. It is clearly the relationship between an addressed location and the decision boundaries that determines the (computational) type of the input.

If this argument has not convinced some readers about the vagueness of the notion of a vector of hidden unit activations having content, then consider the following. Figure 5 shows the label *cow* in the extreme upper left of the space while *condor* is in the extreme lower right. Now suppose we wish to create a type category for words beginning with the letter c. This is quite easy in decision space. We simply add a third dimension and raise both *cow* and *condor* on the z axis. In this case, as shown in Figure 7, we can simply slide in a new decision boundary to separate *cow* and *condor* from the other labelled locations in the space.

So, in what sense can it be that the vectors of hidden unit activation now contain information that condor and cow both belong to the class of words beginning with the letter *c*? OK, so we added a small value on a third dimension. This change does not really affect the distance relationships between the addressed

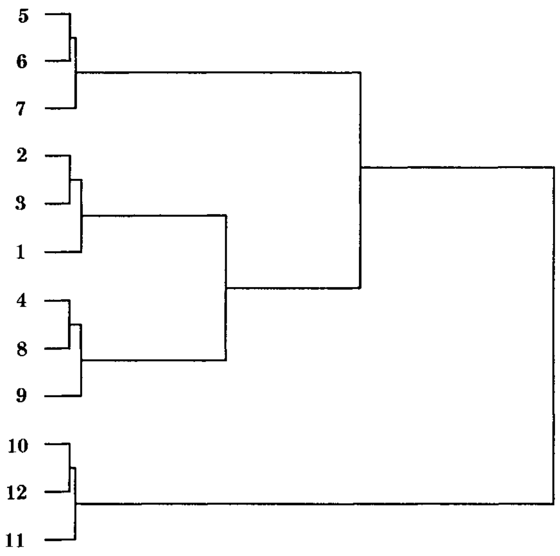


Figure 8 Cluster analysis of the 3-d space shown in Figure 8.

locations, as can be seen from the cluster analysis of the 3-d space, shown in Figure 8, but it would affect their type relationships,

In summary, the examples in this section have shown that it is decision space rather than representation space that indicates the similarities between the inputs. Systematicity arises from the relationships in decision space between each addressed location and the decision boundaries, rather than between the addressed locations themselves. The decision space analysis clearly shows the computational role of the input vectors. The 'meaning' of an input vector in the system is the relation of its address to the decision boundaries of the output units. The idea that the individual addresses share intrinsic content, is only true insofar as inputs with similar addresses may be, depending on the task, more likely to end up in the same region. Any point in a region will output the same as any other point in the same region regardless of distance from the decision boundaries. As the example in Figure 5 shows, two points, 1 and 4 are relatively distant and yet are computationally identical whereas points 4 and 8 are relatively close and computationally disparate. Moreover, we have shown that the address can only be said to contain information about 'type' or 'semantic information' in a vague and unreliable way. This is because the 'type' of an address can be entirely changed just by moving the decision lines, as shown in Figure 6.

4 DISCUSSION

Our aim in this paper was to challenge one of the key assumption of an emerging connectionist theory of mind; that of spatial systematicity. Our approach has been to strip away some of the layers of baggage that could obscure the issue of what and how a connectionist net is computing. This is reminiscent of a similar purge in AI some years ago when McDermott ([15]) railed against the cavalier use of conceptually extravagant, but ultimately meaningless, labels in artificial intelligence work. He complained that the rich web of associations within which each such label is buried both obscured what was really going on among the labelled entities and also led to delusions of grandeur in the experimenters. This hard criticism was generally accepted at the time (although not always acted upon) by the AI community. It is suggested here that this is a healthy step and that it may now be time for all of us 'sub-symbolists' to pay attention to this important lesson. Although it is important not to give up all that has been learned by the Classical tradition, it could well be a fatal distraction to concentrate on satisfying their terminology rather than paying attention to the phenomena that are to be explained.

In our analysis of spatial systematicity, we removed the label *representation* altogether and replaced it with the more neutral terms *address* and *addressed location*. The main problem with the idea of hidden unit distributed representations is that it considers only half of what a net with two layers of weights is doing. Popular neighbourhood analysis techniques such as cluster analysis leave out entirely the role of the output weights. What we have argued here is that because the computational properties of a net and the neighbourhood distances are often correlated, one can be misled into believing that the distance relations are causal in the computation. We set out to dispel this belief by showing, in a number of examples, how the two can be analytically separated. Indeed, it is only the relationship between each addressed location and the collective decision boundaries that causally determines the computation of a net.

The spatial relations between the individual hidden vectors are not a direct analogy to the syntactic relations between Classical representations. Back-propagation works with functional rather than distance similarity among the input vectors. All that happens during learning is that the input and output weights are moved until all of the addressed locations are on the correct sides of all of the decision boundaries. Once this occurs the learning terminates regardless of the specific distances between the points. We have shown here that it is possible for a cluster of addressed locations to have no functional significance for the particular task as shown by the dendogram of the points in Figure 4. The cluster of the numbered points, 4, 8, and 9 had nothing to do with what the net computes according to the decision spaces shown in Figures 5 and 6.

When we show the door to the spatial systematicity assumption two other subsiduary assumptions must also be asked to leave. The first is that connectionist representations (our addressed locations), as vectors of activation, have a systematic internal structure. That is, representations which need to be treated similarly by a given computational process, *are* treated similarly, by virtue of their own internal structure. In one extreme case here (see Figure 8) we showed that although the labels *condor* and *cow* were at extreme corners of the space (almost as far apart as possible) their addressed locations could be treated similarly by the computational processes simply by moving them into a third dimension to enable a decision plane to cut them off from the other addressed locations (yet they could be the same distance or further apart from one another). The second subsidiary assumption is that the internal structure of a representation, which is a function of the content of its superposed constituents, determines its degree of 'semantic similarity' with other compound representations, ie. compound representations with similar structures will have similar 'contents'. Again, the *condor-cow* example, shows this assumption up for what it is. And further, in Figure 5, this assumption would show, in terms of the vocabulary of the spatial systematists, that a *poodle* is 'semantically more similar' to a *snail* than it is to a *fox*, a *wolf* or a *jackal*. The decision space analysis however, reveals a qualitatively different picture, where *poodle, fox, jackal* and *wolf* belong to the same functional category with computational properties distinct from *snail*.

Let us be absolutely clear. A proponent of 'spatial systematicity' may continue to argue that mere addressed location of an input in hidden unit space is still of paramount importance. This is wrong. It is rather the case that location is a relative concept, and it can *only be with respect to the decision boundaries* that an addressed location has computational significance. What we have shown here is that if an input maintains its addressed location, while the decision boundaries are changed, then that input will have a different role in the computation of the network. Indeed, a fixed addressed location for an input may partake in many different computational roles, as a function of the decision boundaries implemented by the output units.

Overall, the arguments presented here suggest that, contrary to the popular practice of concentrating upon a single representational resource in isolation (either the unit or the weight resource), connectionists would do better to adopt a 'whole net' view of representation. Analysing unit activations in isolation is misleading, unless the theorist takes into account decision boundaries implemented by the output units: similarly, analysing decision boundaries without taking into account the addressed locations they partition is meaningless. It remains to be seen how, and in what form, the notion of 'whole net' representation can be fleshed out.

The bottom line is that spatial relations of hidden unit space can be separated *analytically* from the computational roles of the 'representations'. The burden of proof that such a separation cannot occur, either in principle or in practise in specific implementations of cognitive capacities, lies with those who are making the claims about spatial systematicity. Our suggestion is that a more stable cornerstone for a new connectionist theory of mind should be built up on the foundations of decision space presented here.

5 SUMMARY

This chapter was concerned with the underpinnings of the emerging connectionist theory of cognition with regards to the internal workings of connectionist nets (hence the title of the chapter). We did not reconsider the terms of the debate between the Classical and the connectionist schools, and thus, in this sense also, the chapter is an internal report for connectionists. Our aim was to challenge a key assumption of the proposed connectionist representational scheme, namely, that the precise distances between vectors of hidden unit activations, learned by a connectionist net, reflect systematic semantic and/or structural similarity relations. We did not simply attack a *Spatial Strawman* in advancing our argument because, as we have said, current understanding holds that even a *minute* difference in the distance between a given activation vector and any of the others will change the 'meaning' of that vector.

The structure of our argument was in two parts. First, in Section 2, we examined the origins of the Connectionist term 'representation', and described the key 'spatial systematicity' assumption. Second, in Section 3, we presented a detailed argument using constructive methods, drawing upon a simple decision space technique which demonstrated that this assumption may not be warranted except under special circumstances. An important aspect of the argument was to show that the computational role of a, so called, distributed representation may be separated from its specific neighbourhood distance relations. This analysis was then used to support an argument in favour of 'whole net' representation. It may seem to some that we are being a bit 'picky' here, but if we do not make sure that the foundations are correct we could end up with a tower of Babel.

ACKNOWLEDGEMENTS

We would like to thank Lars Niklasson and Timothy van Gelder for helpful comments on an earlier draft of this chapter.

REFERENCES

[1] Aizawa, K. (1992) Review of *Philosophy and Connectionist Theory*. W.Ramsey, S.P.Stich & D.E.Rumelhart (Eds). In *Mind and Language*, 7.

[2] Baldwin, A. (1993) The role of connectionist representation in necessary inference for natural language processing. Unpublished PhD. thesis, University of Exeter.

[3] Chalmers, D.J. (1990) Why Fodor and Pylyshyn were wrong: The simplest refutation. *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, 340-347.

[4] Collins, A.M. & Quillian, M.R. (1969) Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8, 240-247.

[5] Dolan, C.P. & Smolensky, P. (1989) Tensor product production system: a modular architecture and representation. *Connection Science*, 1.1., 53-68.

[6] Elman, J.L. (1989). Representation and structure in connectionist models. *CRL Technical Report 8903*, Center for Research in Language, University of California, San Diego, CA.

[7] Fodor, J.A. (1975) *The Language of Thought*. New York: Crowell.

[8] Fodor, J.A. & McLaughlin, B. (1990) Connectionism and the problems of systematicity: Why Smolensky's solution doesn't work. *Cognition*, **35**, 183-204.

[9] Fodor, J.A., & Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, **28**, 2-71.

[10] van Gelder, T. (1989) Classical questions, radical answers: Connectionism and the structure of mental representations. Horgan,T. & Tienson,J. (Eds) *Connectionism and the Philosophy of Mind.*

[11] van Gelder, T. (1990) Compositionality : A connectionist variation on a classical theme. *Cognitive Science, 14*, 355-384.

[12] van Gelder, T. (1992) Defining 'Distributed Representation'. *Connection Science, 4(3,4)*, 175-192.

[13] Hinton, G.E. (1989) Connectionist learning procedures. *Artificial Intelligence*, **40**, 184-235.

[14] McCulloch W.S. & Pitts W.H. (1943) A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.

[15] McDermott, D. (1976) Artificial intelligence meets natural stupidity. *SIGART Newsletter*, no. 57, 4-9.

[16] Pollack, J.B. (1990) Recursive distributed representations. *Artificial Intelligence*, 46, 77-105.

[17] Rumelhart, D.E., Hinton, G.E. & Williams, R.J. (1986) Learning representations by back-propagating errors. *Nature*, 323, 533-536.

[18] Sharkey, N.E. (1991) Connectionist representation techniques. *AI Review*, 5,3, 143-167.

[19] Sharkey, N.E. (1992) The Ghost in the Hybrid: A Study of Uniquely Connectionist Representations. *AISB Quarterly*. 10-16.

[20] Sharkey, N.E. & Jackson, S.A. (In press) Three horns of the representational trilemma. In V. Honavar & L. Uhr (Eds) *Artificial Intelligence and Neural Networks: Steps towards Principled Integration. Volume 1: Basic Paradigms; Learning Representational Issues; and Integrated Architectures*. Cambridge, MA: Academic Press.

PART III

COMBINED ARCHITECTURES

Part III: Combined Architectures

- Chapter 8 (by Ron Sun) presents a connectionist architecture consisting of two levels: one level for conceptual inference and the other for microfeature-based similarity matching.
- Chapter 9 (by Lawrence Bookman) presents a connectionist architecture that supports both structured and non-structured representations in which knowledge is encoded automatically using information-theoretic methods.
- Chapter 10 (by Charles Lin and Jim Hendler) presents an application of a hybrid system shell.

8

A Two-Level Hybrid Architecture for Structuring Knowledge for Commonsense Reasoning

RON SUN

Department of Computer Science
College of Engineering
The University of Alabama
Tuscaloosa, AL 35487

1 INTRODUCTION

In this chapter, a connectionist architecture for structuring knowledge in vague and continuous domains is proposed. The architecture is hybrid in terms of representation, and it consists of two levels: one is an inference network with nodes representing concepts and links representing rules connecting concepts, and the other is a microfeature-based replica of the first level. Based on the interaction between the concept nodes and microfeature nodes in the architecture, inferences are facilitated and knowledge not explicitly encoded in a system can be deduced via a mixture of similarity matching and rule application. The architecture is able to take account of many important desiderata of plausible commonsense reasoning, and produces sensible conclusions.

One of the most important problems in building intelligent systems (and in modeling cognitive processes) is how to represent knowledge, that is, how to structure knowledge in a way that facilitates common types of inferences to be made. Most commonly, some types of logic or rule-based approaches are adopted ([44] and [13]). However, these approaches are far from matching the capacity and flexibility of human reasoning (see [4] and [39]). What is the problem?

Knowledge is hard to grasp, as discovered by many, including some leading researchers (cf. Minsky [21] and Hayes-Roth et al [13]), notwithstanding the fact that small chunks of knowledge for a narrowly defined domain can be extracted and structured into rule-based, frame-based, or other similar systems.

Psychological experiments reveal that in human cognition various kinds of knowledge exist and they are used in different ways [22]. Many different types of inferences can be performed in commonsense reasoning, sometimes based on the same set of knowledge [20]. In addition, most commonsense knowledge are uncertain, fuzzy, or probabilistic (cf. [23], [41], and [6]).

In view of these problems, in order to build intelligent systems that are more capable than the existing ones in producing flexible, plausible and useful inferences, there is clearly a need to sharpen up means for representing knowledge; that is, there is a need to structure knowledge in ways that maximize the inferential capability while minimizing the inferential complexity (the *performance* issue does need to be considered). This is a tough goal to achieve, since the two subgoals are apparently mutually contradictory. (For some domains that will be looked into here, a nice balance between the two subgoals is indeed achievable.)

Some attempts at providing more flexibility to intelligent systems result in some partially successful models for representing knowledge and/or performing reasoning. One such model is fuzzy logic [43], which allows vague concepts and deals with vagueness in reasoning. It envisages a concept as a set of objects each of which belongs to the set to a certain degree, as measured by a *grade of membership*. Thus, vague concepts are accommodated and objects may belong to a particular concept *partially*, without the need of forcing a dichotomatical true/false decision. (This work will adopt this view of concepts as partially true descriptions of objects, but will not adopt the logical operations defined on top of that, i.e. MIN/MAX, as in [43]).

Beside fuzzy logic, there are a number of other rule-based approaches, for example, probabilistic reasoning [23], Dempster-Shafer evidential reasoning [26], and so on, each of which is good at capturing a particular aspect of vagueness of real world knowledge. However, they are not *meant* to deal with aspects other than the one for which they are specifically designed.

Another popular approach is that of PDP models (see [25] and [1]). Using networks of simple processing elements, typically with a global multilayer feedforward structure and with a continuous, sigmoidal activation function in each processing element, these models are capable of virtually any continuous mapping [16], including dealing with flexible reasoning based on similarity as mappings between two sets of concepts. This purely similarity-based, generic framework can be applied to a large variety of knowledge intensive tasks, by encoding knowledge distributively in link weights acquired from applying learning algorithms. However, these models lack certain basic characteristics for flexible yet precise commonsense reasoning, namely, explanation, symbolic processing, interaction with humans in acquiring and modifying knowledge, interpretation of internally stored knowledge, and handling of discontinuous cases (see [33]; more on this later).

In the discussion here, let us generally assume that knowledge in a domain is composed of knowledge *statements*, such as Horn clauses, which could be atomic propositions or rules consisting of a number of atomic propositions expressing antecedents and consequents respectively. This is an acceptable assumption, because almost all knowledge-based systems are in such forms (or can be transformed into such forms, for example, frames, scripts, etc.; see [12]). (One exception is neural networks, which encode knowledge in a set of numerical weights, which are difficult to interpret in terms of rules. One of the goals of this work is to show that there is an alternative to the black-box style neural network models, and the architecture proposed here not only serves as an alternative but also has some important advantages.) From here on, when we discuss a domain, we will think of it as composed of a *space* of *primitive statements* (i.e., the set of all possible primitive statements), and the main operation in that space is *match*. When two statements are deemed matching each other, inferences are enabled by associating knowledge of one statement with the other. For example, suppose there is the following space of primitive statements: $\{a, b, c, d, e, f\}$, and there are the following associated compound knowledge statements: $c \longrightarrow d$ and $e \longrightarrow f$. When given a, matching statements are searched for. Suppose e matches a (exactly or approximately), then the knowledge statement $e \longrightarrow f$ can be applied, and f can be deduced (exactly or approximately).

The present work will be mainly concerned with the *vague* domains. By vague domains, it is meant the type of domains that are composed of inexact statements (fuzzy, probabilistic, etc.), and the match between two statements in a domain is not all-or-nothing: a continuum can be formed that ranges from perfect (exact) match to no match (irrelevance) to the exact opposite (the negative perfect match). The key features of such domains are (1) inexactness, allowing partial match situations, and (2) continuity, with varying degrees of partial match.

Given the above, it is impossible to have all the knowledge statements that can cover all possible situations in such domains: the number of possible situations can be infinite, because a continuum can exist going from one statement (any particular statement) to the statement that is the exact opposite. Given this infinite space, knowledge statements need to be devised and structured economically, in ways that can cover a domain as accurately as possible within the constraint of resources. The question of how to structure knowledge to help to guide and facilitate reasoning in a domain also needs to be considered.

In the rest of this chapter, some discussion of vagueness will be provided, and then a two-level structuring of knowledge for vague domains will be proposed. Based on a particular set of requirements for plausible inferences, the parameter values of a two-level architecture satisfying these requirements will be derived. A set of experiments will also be presented to further illustrate the architecture. Finally, brief comparisons and conclusions will complete the chapter.

2 DEVELOPING A TWO-LEVEL ARCHITECTURE

2.1 SOME ANALYSES

Let us consider how we can better structure our knowledge, given the aforementioned considerations. First of all, we need to have explicit knowledge statements in our system. This is because explicitly stored knowledge statements can provide clarity, modularity, human comprehensibility, and explanations needed for interaction. We simply choose *Horn clauses* as the form of such knowledge statements (see [3]). In its simplest case, a Horn clause just states a proposition or a concept, without any pre-conditions or possible consequences. In more complex cases, a Horn clause is a rule consisting of a number of antecedents, which are simple propositions/concepts, and a consequent, which is also a simple proposition/concept. We choose Horn clause because of the simplicity of the formalism, its popularity in knowledge-based systems, its expressive power, and its inferential efficiency [3].

Second, we need to have a better grasp of the various kinds of vagueness in domains. For one thing, there should be a degree of *confidence* or *certainty* associated with each primitive knowledge statement (or its instantiation), judging how close, and/or how likely an instance of that statement is to conform to the ideal of the concept involved (see [41]). For example, "warm areas are suitable for rice growing" can be expressed in the rule

$$warm \longrightarrow rice\text{-}growing$$

Here "warm" is not a crisp concept (knowledge statement); various degrees of "warm" exist. We should allow a degree of confidence to be associated with the knowledge statement. We can accept as input to a system an instance of the knowledge statement of a place being "warm" with an associated confidence degree. We then match it with an existing rule (or compound knowledge statement), to form an instantiation of the rule. And we derive the conclusion knowledge statement of a place being "rice-growing" with a corresponding degree of confidence determined by, among other things, the degree of confidence of being warm.

We also need a way of associating a confidence value with a rule itself, determining how likely it will hold. Moreover, various vague evidence leading to the same conclusion should be summed up; that is, the degrees of confidence of different pieces of evidence in a rule have to be *accumulated* in some way, for example, by a weighted-sum. An example is as follows,

$$subtropical\ rainy\ flat\ evergreen\ \neg flood \longrightarrow rubber\text{-}producing\text{-}area$$

That is, an area with subtropical and rainy weather, flat terrain, and evergreen vegetation cover but no frequent flood tends to be an rubber-producing area. If one only knows that an area is subtropical, rainy, and without flood, one may conclude that it could be a rubber-producing area; if one knows that an area is subtropical, rainy, with flat terrain, and without flood, one can conclude that it is more likely to be a rubber-producing area; if one knows all of the above conditions, one can conclude that the likelihood is very high. A cumulative evidentiality is in working here, which should be taken into consideration.

There is yet another type of vagueness, which can be further pinned down. Basic concepts (or primitive knowledge statements) are *similar* to each other to varying degrees. Even though something is best described by one particular knowledge statement, other statements may also apply. Especially when there is no inference that can be made with the best matched case, turning to other related cases will definitely be of help. This is in a way similar to analogical reasoning (or case-based reasoning), in that some seemingly different knowledge is brought together from a partial match of two chunks of knowledge. In some sense, we need a rudimentary form of analogical reasoning capability in dealing with vague domains. An example from the commonsense geographical reasoning domain is as follows: "Columbia basin" is described as tropical river-basin and coastland, and "Ecuador coast" is described as a tropical, coastal lowland with rainforest cover. In order to deduce possible agricultural products of "Columbia basin", we notice its similarity to "Ecuador coast" in terms of their geographical features. Since the latter produces (among other things) bananas, we can conclude that the former may produce bananas too. For another example, "Northern Brazil" is described as a tropical, hilly plateau with rainforest, and "Bolivia oriente rainforest area" is described as tropical plain and lowland with rainforest. The former is a rubber-producing area. Because of the feature similarity, we might conclude that the latter is likely to be one too.

To sum up, some precisely specified rules are needed, and in addition to such rules, vagueness in a domain needs to be dealt with by utilizing continuous numerical evidential combination and similarity-based inferences. Similarities can be explored based on features (or microfeatures, as used in [25]) to provide a fuller coverage of all possible situations in a domain.

2.2 A TWO-LEVEL ARCHITECTURE

A two-level approach for structuring knowledge to take care of the two types of vagueness was proposed in [34]. We call this architecture CONSYDERR, which stands for *a CONnectionist SYstem with Dual-representation for Evidential Robust Reasoning*. One level of this architecture is the *concept level*, which contains primitive knowledge statements, or concepts. This level consists of a collection of *nodes*, or processing elements, for representing the concepts in the domain. For expressing compound knowledge statements, or rules, these nodes are connected via links from antecedents of a rule to consequents of the rule. (This level is the top level in Figure 1.) The other level (the bottom level) is the *microfeature level*, which contains nodes each of which represents a fine-grained element (a microfeature) in the meanings of the concepts represented in the top level. Each node in the top level is connected to all the relevant microfeature nodes in the bottom level; once a concept node is activated, the related microfeature nodes will be activated subsequently from inter-level connection, and vice versa. Links in the top level are replicated diffusely in the microfeature level by multiple links between two sets of microfeature nodes representing an antecedent and a consequent of a rule respectively. The first type of vagueness is handled by utilizing weighted-sum computation in each nodes, which is a continuous mapping, accumulating evidence from different sources (see [33, 35]), in computing its output activation. It is proven that such a function can actually implement Horn clause logic as a special case [35]. The second type of vagueness is handled by a *similarity matching process* based on microfeatures. In this structure, similarity matching is accomplished through the interaction between the two levels, by a top-down/settling/bottom-up cycle (see Figure 1).

Equations for the computation of the three phases are specified as follows:

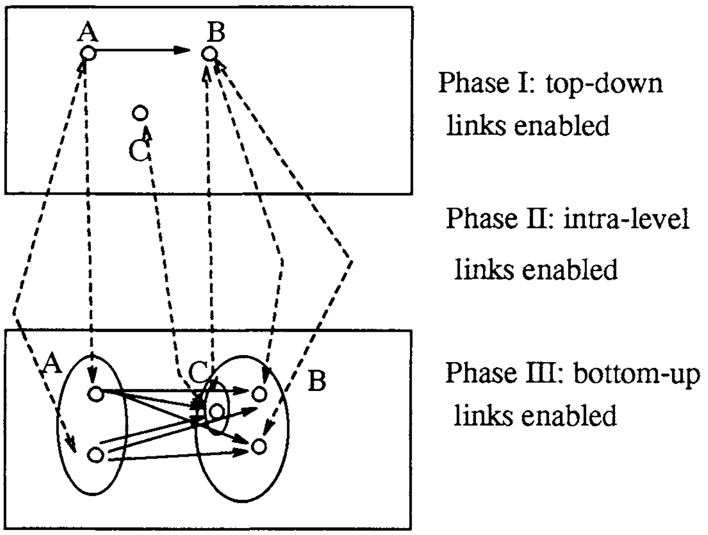

A
B
C
A
C
B
Phase I: top-down links enabled
Phase II: intra-level links enabled
Phase III: bottom-up links enabled

Figure 1 The Two Level Structure

For the top-down phase,

$$ACT_{x_i}(t+1) = \max_A(td_A * ACT_A(t))$$

where ACT is the activation value of a node and A is any node in the top level that has $x_i \in F_A$ (the set of microfeatures connected to A); *td* is a weight (to be determined later). That is, a microfeature node receives activation from the corresponding concept nodes, and chooses the largest value.

For the settling phase, in the top level

$$ACT_C(t+1) = \sum_i r_i * ACT_{A_i}(t)$$

and in the bottom level

$$ACT_y(t+1) = \sum_j lw_j * ACT_{x_j}(t)$$

where r's and lw's are link weights (representing rule strengths), and lw's can be determined from corresponding r's (as will be shown later); A_i's and x_i's are the activations of nodes that are related respectively to C and y's by links (rules). That is, in this case, each node receives activations from other nodes at the same level (which are related to it by rules) and does a weighted-sum for computing its own activation.

For the bottom-up phase,

$$ACT_C(t+1) = max(ACT_C(t), \sum_{y_i \in F_C} bu_C * ACT_{y_i}(t))$$

where C is any top level node, and y_i's are its corresponding microfeature nodes (because $y_i \in F_C$); *bu* is a weight (to be determined later). That is, a concept node receives activation from its corresponding microfeature node, and chooses the value as its activation if it is greater than its original activation.

When applying this cycle, first some nodes in the top level get activated by external inputs (and clamped). Then the top-down phase will activate (and clamp) the microfeature nodes corresponding to the active concept nodes. In the settling phase, links representing rules related to those activated nodes take effect in both levels. Concepts may have overlapping microfeature representations because they share some common microfeatures due to similarity; so some of the microfeature representations of concepts will be partially activated if a concept *similar* to them is activated (in the bottom level). Finally in the bottom-up phase, fully or partially activated microfeature representations will go back up to activate the corresponding nodes in the top level. The result can be read off from the top level.[1]

Notice the massive parallelism in the above architecture: activations are propagated, in a massively parallel fashion, from all pre-link nodes to all post-link nodes; each node receives inputs as soon as it can, and therefore fires as soon as it can, ensuring a maximum degree of parallelism in terms of rule application. In terms of similarity matching, all similar concepts are activated (in their microfeature representations) immediately once an original concept is activated, and simultaneously matched with the original one (through top-down and bottom-up flows); thus the architecture is extremely efficient by employing the two levels. The parallelism in this architecture accounts well for the similar parallelism and spontaneity in human reasoning processes as identified in, for example, Collins & Michalski [4] and Sun & Waltz [33].[2]

[1]Each node in the system has one or more sites (cf. [7]), each of which computes the weighted-sum (or any other similar functions whenever needed) of the inputs. The maximum of the values computed by all the sites is taken to be the activation value of the node.

[2]One problem that is not addressed here is the variable binding problem, that is, how arguments can be associated with predicates in inference. This is especially difficult for connectionist

3 FINE-TUNING THE STRUCTURE

3.1 BASIC DESIDERATA

To show that the two-level architecture proposed above is versatile enough to accommodate special requirements that are often associated with various kinds of reasoning tasks, we will see how a set of desiderata for dealing with the geographical commonsense reasoning tasks (see [34]) can be used to determine the parameters of the architecture.

All of the desiderata and the requirements for some commonsense reasoning in geographical domains are determined in [34] and they can be summarized together as follows (divided into three categories: rule application, similarity matching, and inheritance):

Similarity. A similarity measure s_{AB} measures the similarity between A (the target) and B (the source), namely, "$A \sim B$".[3] It is needed because in vague domains we can and must reason based on similarities of knowledge statements, in order to reach plausible conclusions with an incomplete knowledge base. A similarity measure has the following requirements (see [34] and references cited therein for detailed justifications; they are too long to replicate here):

> $s_{AB} \propto |F_A \cap F_B|$, that is, the similarity between two concepts is proportional to the amount of their microfeature overlapping.
>
> $s_{AB} \propto \frac{1}{|F_B|}$, that is, the similarity is inversely proportional to the number of microfeatures B has, when everything else is equal.
>
> $s_{AB} \not\propto \frac{1}{|F_A|}$, that is, the similarity is not (inversely or not) proportional to the number of microfeatures A has, when everything else is equal.

networks (when used as implementational means), because of the simplicity and homogeneity of such networks. Nevertheless, such networks have been shown to handle variable binding to a large extent (see Sun [35] for details).

[3] Here similarity from A to B means that, when there is no direct knowledge about the concept A available, the concept B, which is similar to A, can be utilized to find plausible answers. This situation, which can also be termed similarity-based induction, is different from the generic, context-free notion of similarity.

Rules. Rules are needed for expressing precisely the knowledge that a system does possess. Coupling such precise knowledge statements with similarities, a lot of plausible inferences can be made. The following cases of rules and mixed rules/similarities have respective requirements:[4]

(1) A $\longrightarrow$ B: if A is activated, then $ACT_B = r_{AB} * ACT_A$, where ACT_A is the activation value of A, ACT_B is the activation value of B, and r_{AB} is the strength of the rule between A and B (the same below);

(2) A $\sim$ B, B $\longrightarrow$ C: if A is activated, then $ACT_B = s_{AB} * ACT_A$, and $ACT_C = r_{BC} * ACT_B$, where s_{AB} is the similarity between A and B (the same below);

(3) A $\longrightarrow$ B, B $\sim$ C: if A is activated, then $ACT_B = r_{AB} * ACT_A$, and $ACT_C = s_{BC} * ACT_B$;

(4) A $\longrightarrow$ B, B $\longrightarrow$ C: if A is activated, then $ACT_B = r_{AB} * ACT_A$, and $ACT_C = r_{BC} * ACT_B$;

(5) A $\longrightarrow$ B, B $\longrightarrow$ C, C $\longrightarrow$ D: if A is activated, then $ACT_B = r_{AB} * ACT_A$, $ACT_C = r_{BC} * ACT_B$, and $ACT_D = r_{CD} * ACT_C$;

(6) A $\sim$ B, B $\longrightarrow$ C, C $\longrightarrow$ D: if A is activated, then $ACT_B = s_{AB} * ACT_A$, $ACT_C = r_{BC} * ACT_B$, and $ACT_D = r_{CD} * ACT_C$;

(7) A $\longrightarrow$ B, B $\sim$ C, C $\longrightarrow$ D: if A is activated, then $ACT_B = r_{AB} * ACT_A$, $ACT_C = s_{BC} * ACT_B$, and $ACT_D = r_{CD} * ACT_C$;

(8) A $\longrightarrow$ B, B $\longrightarrow$ C, C $\sim$ D: if A is activated, then $ACT_B = r_{AB} * ACT_A$, $ACT_C = r_{BC} * ACT_B$, and $ACT_D = s_{CD} * ACT_C$;

(9) A $\sim$ B, B $\longrightarrow$ C, C $\sim$ D: if A is activated, then $ACT_B = s_{AB} * ACT_A$, $ACT_C = r_{BC} * ACT_B$, and $ACT_D = s_{CD} * ACT_C$.

Inheritance. Inheritance is inference based on knowledge (statements) associated with a superclass or a subclass of a concept. This problem is important, because different concepts (primitive knowledge statements) may bear relationships to each other as superclass/subclass. Therefore it is necessary to consider their mutual interaction, and to organize knowledge around an inheritance hierarchy for storage economy. Let A $\supset$ B, and therefore $F_A \subset F_B$; that is, the larger the extension the smaller the intension (the microfeature set); if A is a superset of B, the intension (the microfeature set) of A is the subset

[4] We only deal with rules with only a single premise here; rules with multiple premises are just extensions of these cases (see [35] for a detailed treatment). Note also that, unlike symbolic systems, here various sorts of chaining have to be dealt with on a one-by-one basis.

of the intension (the microfeature set) of B (see [19] for explanations). The following cases (taken from [36]) should be checked:

(1) A has a property value C, and B has no specified property value. If B is activated, then C should be activated.

(2) B has a property value D, and A has no specified property value, if A is activated, D should be activated too.

(3) A has a property value C and B has a property value D $\neq C$. if A is activated, C should win over D.

(4) A has a property value C and B has a property value D $\neq C$. if B is activated, D should win over C.

(5) A has a property value C which has a feature set F_C, and B has a property value $D \supset C$ $(F_D \subset F_C)$. If A is activated, C should win over D.

(6) A has a property value C which has a feature set F_C, and B has a property value $D \supset C$ $(F_D \subset F_C)$. If B is activated, D should win over C.

With all these constraints established, we can proceed to derive the exact specifications of the parameters, including top-down weights (denoted as td), bottom-up weights (denoted as bu), and weights (denoted as lw) for links between two microfeature nodes which diffusely replicate the rule links r (see Figure 2). **This architecture can solve the similarity matching problem, the rule application problem, and the inheritance problem, with the same set of (appropriately set) parameters.** [5]

3.2 PARAMETERS DERIVATION

Below we will analyze the requirements and derive the parameters. For the sake of simplifying the discussion, we assume in the following discussion that the original rule strengths in the top level are all the same, that is, the maximum value 1, We first direct our attention to inheritance/cancellation. Consider cases three and four. When A is activated (but not B), we want C to be activated more strongly than D in the bottom level. Then during the bottom-up process,

[5]We will use continuous activation values (for representing confidence values), approximately between -1 and 1, in which 1 represents full confidence, 0 represents unknown, and -1 represents full negative confidence. We will not use thresholds for the simple reason that we adopted continuous activation values which represents continuous confidence values.

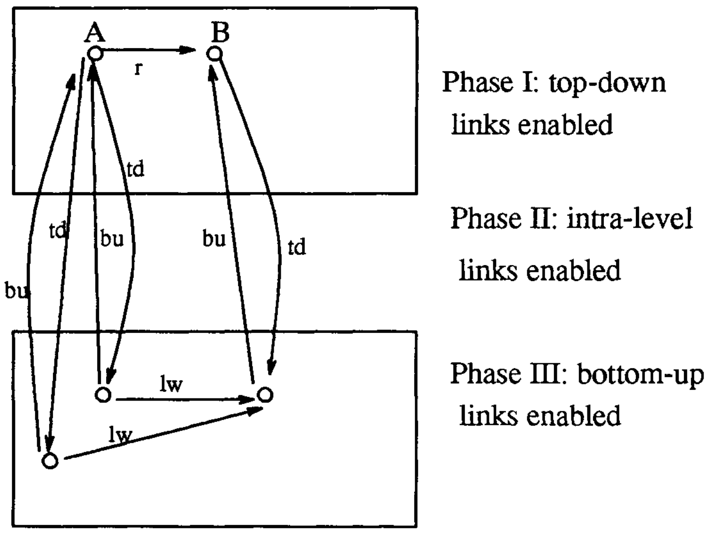

A
B
r
td
bu
bu
td
td
bu
lw
lw
Phase I: top-down links enabled
Phase II: intra-level links enabled
Phase III: bottom-up links enabled

Figure 2 A Generic Model

these activation values will be transmitted to the corresponding concept nodes. To make sure that C is activated more strongly than D, *lw* (weights on the links that diffusely replicate the link in the top level) should be somehow inversely related to the size of the microfeature set of the originating concept. Assume the original link weight is r, and the weights for links in the bottom level that replicate (diffusely) the original link are (uniformly) lw. Let A be the source node, and F_A be its microfeature set. Similarly, let C be the destination node, and F_C be its microfeature set.

$$lw_{AC} = \frac{r}{f(|F_A|)}$$

for any A and C, where f is a monotonic increasing function, linear or otherwise. Similarly, when B is activated (but not A), we want D to be activated more strongly than C in the bottom level. Because the microfeature set of A is a subset of that of B (as explained before), the total activation transmitted to C or D should be related to the sizes of the respective microfeature sets. Otherwise C and D will receive the same amount of activation, and therefore it will become impossible to differentiate the two. Since the total activation is equal to the size of the microfeature set of the originating concept times the activation transmitted along each individual link, to make sure that D is activated more strongly than C in this case, we must make f sublinear, so that the total activation transmitted will be related to the size of the microfeature set of the originating concept.

It is easy to confirm that no matter what bu and td are used, with this lw function, C and D in the bottom level (F_C and F_D) will have the right activation in both cases. The details follow: suppose A is activated, and λ is the activation of the microfeature nodes of A due to top-down activation, and σ is the bottom-up weight (the same for both C and D, because they both have only one microfeature node):

$$\begin{aligned} ACT_C &= \sigma \sum_{F_A} lw_{AC} \lambda \\ &= \sigma |F_A| \frac{r}{f(|F_A|)} \lambda \end{aligned}$$

$$\begin{aligned} ACT_D &= \sigma \sum_{F_A} lw_{BD} \lambda \\ &= \sigma |F_A| \frac{r}{f(|F_B|)} \lambda \end{aligned}$$

so C is activated more strongly than D. In the other case, if B is activated (with activation value λ; the same for its microfeature nodes),

$$\begin{aligned} ACT_D &= \sigma \sum_{F_B} lw_{BD} \lambda \\ &= \sigma |F_B| \frac{r}{f(|F_B|)} \lambda \end{aligned}$$

$$\begin{aligned} ACT_C &= \sigma \sum_{F_A} lw_{AC} \lambda \\ &= \sigma |F_A| \frac{r}{f(|F_A|)} \lambda \end{aligned}$$

so D is activated more strongly than C.

We are now ready to examine cases five and six, which are more complicated: because F_C is embedded in F_D (because C is a superclass of D, as explained before), it is imperative that we pick the right bu function that takes into account all effects, desirable or undesirable, of sizes of microfeature sets. Look at case five. In order to have C activated more strongly than D,[6] we have to take

[6]Let us assume that there is nothing going on in the top level, and all activations come bottom-up.

into account the sizes of the microfeature sets of C and D (i.e., F_C and F_D) in determining bu. And bu should be inversely related to the size of the microfeature sets of the node in the top level with which the particular bu is associated. Assume all microfeatures of C and D are activated to the same degree, and F_C is embedded in F_D, that is, C has fewer microfeatures than D; if we have a uniform bu (equal to some σ), we will have an incorrect result (D being more strongly activated than C):

$$\begin{aligned}
ACT_C &= \sum_{F_C} bu_C * (activation\ \ of\ \ each\ \ node\ \ in\ \ F_C) \\
&= \sum_{F_C} bu_C \sum_{F_A} \lambda * lw_{AC} \\
&= |F_C| * \sigma * |F_A| * \lambda * \frac{r_{AC}}{f(|F_A|)} \\
&= |F_C| * \sigma * \lambda * r_{AC}
\end{aligned}$$

provided f is the identity function, where λ is the activation of the microfeature nodes of A, which are all the same;

$$\begin{aligned}
ACT_D &= \sum_{F_D - F_C} bu_D (activation\ \ of\ \ each\ \ node\ \ in\ \ F_D - F_C) + \\
&\quad \sum_{F_C} bu_D (activation\ \ of\ \ each\ \ node\ \ in\ \ F_C) \\
&= \sum_{F_D - F_C} bu_D \sum_{F_A} \lambda * lw_{BD} + \sum_{F_C} bu_D \sum_{F_A} \lambda * lw_{AC} \\
&= |F_D - F_C| * \sigma * |F_A| * \lambda * \frac{r_{BD}}{f(|F_B|)} + |F_C| * \sigma * |F_A| * \lambda * \frac{r_A C}{f(|F_A|)} \\
&= \sigma * \lambda * r_{AC} * |F_A|(|F_C| * \frac{1}{|f(F_A|)} + |F_D - F_C| * \frac{1}{f(|F_B|)}) \\
&= \sigma * \lambda * r_{AC}(|F_C| + |F_A| * |F_D - F_C| * \frac{1}{|F_B|})
\end{aligned}$$

provided that f is the identity function and $r_{AC} = r_{BD}$. Comparing the two formulas, clearly $D > C$, which is wrong. On the other hand, if we make bu to be inversely proportional to the size of the microfeature set of the CL node with which the bu is associated, we will have $C > D$, which is correct. So what we can do now is simply to have a function, g, that is faster than a constant (with coefficients properly adjusted to guarantee that the asymptotic properties also hold for small values), and to let

$$bu_C = \frac{1}{g(|F_C|)}$$

for all C. Examples of such g include $\sqrt{x}$, $log\,x$, the identity function, or other linear functions, with coefficients equal to 1.

In case six, we want the opposite: $D > C$. We can perform a similar analysis. In this case, we would rather have little or no influence from the sizes of the microfeature sets. It is easy to see why: If we assume that $bu = \sigma$ (i.e., $g(x) = \frac{1}{\sigma}$), then, derived the same way as before,

$$\begin{aligned} ACT_C &= \sum_{F_C} bu_C \sum_{F_A} \lambda * lw_{AC} \\ &= |F_C| * \sigma * |F_A| * \lambda * \frac{r_{AC}}{f(|F_A|)} \\ &= |F_C| * \sigma * \lambda * r_{AC} \end{aligned}$$

and

$$\begin{aligned} ACT_D &= \sum_{F_D} bu_D \sum_{F_B} \lambda * lw_{BD} \\ &= |F_D| * \sigma * |F_B| * \lambda * \frac{r_{BD}}{f(|F_B|)} \\ &= \sigma * \lambda * r_{AC} * |F_B| * |F_D| * \frac{1}{f(|F_B|)} \\ &= \sigma * \lambda * r_{AC} * |F_D| \end{aligned}$$

provided f is the identity function and $r_{AC} = r_{BD}$, where λ is the activation of the microfeature nodes of A, which are all the same. Comparing the two formulas, clearly $D > C$, which is correct. But if we try to have a linear function or a function that is faster than linear functions, it can be easily shown that we will not get the correct result.

Combining results from the above two cases, we conclude that g, as part of bu, has to be a function that is slower than linear functions, but faster than constants.

Although the above derivation assumes that $f(x) = x$, as we have shown before, $f(x)$ has to be slower than linear. To right the situation, we just have to make $f(x)$ as close to linear functions as possible, so that the non-linearity of $f(x)$ will not affect the obtained relation ($C > D$ or $D > C$), given the ranges of $|F_A|$, $|F_B|$, $|F_C|$, and $|F_D|$. For example, we can choose $f(x) = x^{999/1000}$ and $g(x) = x^{9/10}$.

For the first two cases of inheritance, they can be handled by a mixture of rule application and similarity matching. Case one can be described as $B \sim$

$A, A \longrightarrow C$, and can be handled as mixed rule application and similarity matching. As will be analyzed later, if B is activated, then

$$ACT_C = ACT_B * s_{BA} * r_{AC}$$

Case two can be described as $A \sim B, B \longrightarrow D$, and, as will be shown later, if A is activated, then

$$ACT_D = ACT_A * s_{AB} * r_{BD}$$

Let us look into the similarity cases. Given the basic desiderata for similarity, we can think of many different measures (cf. [38], [22], and [11]), such as

$$s_{AB} = f1(|F_A \cap F_B|) - f2(|F_A - F_B|) - f3(|F_B - F_A|)$$

that is, the contrast model of Tversky [38]. Or

$$s_{AB} = \frac{f1(|F_A \cap F_B|)}{f2(|F_A - F_B|) + f3(|F_B - F_A|)}$$

that is, the ratio model of Tversky [38]. Yet others include

$$s_{AB} = \frac{f1(|F_A \cap F_B|)}{f2(|F_A|) + f3(|F_B|)}$$

$$s_{AB} = \frac{f1(|F_A \cap F_B|)}{f2(|F_A|) * f3(|F_B|)}$$

$$s_{AB} = \frac{f1(|F_A \cap F_B|)}{f3(|F_B|)}$$

Many more models can be constructed. However, when we measure them against our previous desiderata, only the last one is acceptable, because it does not involve F_A.

Looking at the matter from a different perspective, considering the implementational issues, we want as simple a formula as possible, not in terms of numbers of parameters or the time complexity of computation, but in terms of ease of implementing it in a connectionist fashion with a set of simple, autonomous, locally connected nodes. We want (1) all computation to be local, (2) only simple messages to be passed around, and (3) no extra nodes to be added (see Feldman [7] for similar points). With these three criteria in mind, again only the last model can be selected (details are omitted).

Suppose A is externally activated, then because of the microfeatures shared with A, B will later be activated: the activation of A will first go top-down to its microfeature nodes through weights td; due to overlapping, some of the microfeature nodes of B (in $F_A \cap F_B$) will be activated; then at the bottom-up phase, the activation of the microfeatures of B goes up to the node B in the top level, through weights bu ($= \frac{1}{g(|F_B|)}$). According to what we derived so far,

$$ACT_B = td_A * ACT_A * |F_A \cap F_B| * \frac{1}{g(|F_B|)}$$

To make B match what is obtained from a similarity measure, specifically,

$$s_{AB} = \frac{|F_A \cap F_B|}{|F_B|}$$

we have to choose td as

$$td_A = 1$$

and we have to choose g as close to the identity function as possible, in order to make $\frac{ACT_B}{ACT_A} \approx s_{AB}$. Thus, we determine yet another parameter.

Now we shall check to see if the parameters derived so far satisfy the requirements for correct rule application and mixed similarity matching/rule application (including the first two cases of inheritance). Let us verify them:

(1) For A $\longrightarrow$ B, if A is activated, then $ACT_B = r_{AB} * ACT_A$ in the top level, where r_{AB} is the weight on the link between A and B (the same below), and the bottom-up activation is

$$ACT_A * |F_B| \frac{|F_A| \frac{r_{AB}}{f(F_A)}}{g(F_B)} \approx r_{AB} * ACT_A$$

So the overall result is $ACT_B \approx r_{AB} * ACT_A$;

(2) For A $\sim$ B, B $\longrightarrow$ C, if A is activated, then

$$\begin{aligned} ACT_B &= \sum_{F_C} \frac{\sum_{F_B \cap F_A} ACT_A \frac{r_{BC}}{g(F_B)}}{g(F_C)} \\ &= \frac{|F_C|}{g(F_C)} ACT_A r_{BC} \frac{|F_B \cap F_A|}{g(F_B)} \\ &\approx ACT_A * s_{AB} * r_{BC} \end{aligned}$$

(3) For A $\longrightarrow$ B, B $\sim$ C, if A is activated, then

$$\begin{aligned} ACT_C &= \sum_{F_B \cap F_C} \frac{\sum_{F_A} ACT_A \frac{r_{AB}}{g(F_A)}}{g(F_C)} \\ &= \frac{|F_B \cap F_C|}{g(F_C)} ACT_A r_{AB} \frac{|F_A|}{g(F_A)} \\ &\approx ACT_A * r_{AB} * s_{BC} \end{aligned}$$

All the other cases, (4), (5), (6), (7), (8), and (9), can be verified the same way.

4 EXPERIMENTS

4.1 REASONING WITH GEOGRAPHICAL KNOWLEDGE

Let us look into reasoning with geographical information. Utilizing the two-level idea, the representation of the geographical knowledge is divided into two categories: concepts (primitive knowledge statements), which include basic geographical areas and regional characterizations (such as "cattle-country"), and microfeatures, which include basic geographical descriptions of areas, such as "highland", "mountainous", and "tropical", etc. Concepts are represented in the top level, and microfeatures are represented in the bottom level. Each area is connected to concepts describing its agricultural products by rules, implemented as links. Each geographical area represented in the top level is connected to its corresponding microfeatures in the bottom level, and because of the fact that microfeatures are shared by similar concepts, the microfeature representation is similarity-based, i.e., two concepts have overlapping microfeature representations if and only if the two are similar and the amount of overlapping is proportional to the degree of the similarity between them, as alluded to before. Thus, inferences are enabled through similarity matching, and a fuller coverage of the domain is ensured.

Some of the data stored in the system are tabulated: Figure 3 lists some of the geographical areas included in the system, most of which are in South America; Figure 4 lists concepts for characterizing a geographical area in terms of its agricultural products, such as rice-growing-area, cattle-country, etc.; Figure 5 lists microfeatures used. The fact that the system is fairly large ensures that the experiment is meaningful.

"Chaco"	"Honduras"
"Uruguay-coastal"	"Uruguay-plateau-highland"
"Mendoza"	"Llanos"
"w-Peru"	"c-Peru"
"e-Peru"	"Bolivia-orient-grassland"
"Bolivia-orient-rainforest"	"Bolivia-cordillera-occidental"
"e-Paraguay"	"w-Paraguay-forest"
"w-Paraguay-savanna"	"Panama-lowland"
"w-Texas"	"Guiana-pgs"
"Guiana-hilly-country-forest"	"Guiana-hilly-country-savanna"
"Guiana-plain"	"Bolivia-SW-highlands"
"Brazil-cw"	"Brazil-s"
"Brazil-saopaulo"	"Brazil-e"
"Brazil-ne"	"Brazil-n"
"Chile-n"	"Chile-s"
"Chile-c"	"Argentina-ne"
"Argentina-pampa"	"Argentina-Patagonia"
"Argentina-andeanhighland"	"Columbia-w"
"Columbia-e"	"Columbia-basin"
"Ecuador-coast"	"Ecuador-highlands"
"Venezuela-Llanos"	"Venezuela-coastalplain"
"Suriname-coastalplain"	"Suriname-plateau"

Figure 3 Geographical Regions Included in GIRO

It should be stressed that the process of knowledge acquisition for this system is straightforward and systematic: nothing is tuned arbitrarily just for getting one outcome or the other. Specifically, the knowledge in the system is obtained from encyclopedias, such as *Encyclopedia Britannica* or *Encyclopedia Americana*, in the form of a basic geographical region (a region with relatively uniform characteristics), its products, and its geographical features. These types of information is well documented and rather extensive in source books.

In extracting information from source books, there are some subtleties that have to be taken into consideration. Each article regarding a particular region is written by a particular researcher familiar with that region, and varies in depth, presentation, amount of details and emphasis. This diversity inevitably has adverse effects on the accuracy of the specification. The problem is the lack of details on the one hand and too much detail on the other hand. When there are not enough details from one sourcebook, we can find another sourcebook and try to fill in what is needed. In case of too much detail, we have to be

"cotton-producing-area"
"coffee-growing-area"
"wine-producing-area"
"potato-growing-area"
"rubber-producing-area"
"goats-area"
"rice-growing-area"
"wheat-growing-area"
"soybean-growing-area"
"rubber-producing-area"
"sheep-country"
"producing-banana"
"producing-tropical-fruits"
"corn-growing-area"
"sugar-producing-area"
"fruit-veg-growing-area"

Figure 4 Regional Characterization Included in the System

temperate	arctic	woodland	plain	mediterrainian
plateau	Mts	coastal-land	lake	tropical
lowland	hill	river-valley-basin	swamp	rainforest
evergreen	deciduous	highland	upland	sparsely-populated
densely-populated	fertile	infertile	flood	prairie
dependable-rainfall	scrub	farming	rugged	subtropical
rainy	savanna	dry-arid	grassland	desert

Figure 5 Geographical Features Included in the System

very careful in selecting the most important and relevant information out of the tangled web of irrelevant descriptions. As a rule of thumb, we usually disregard information associated with phases such as "plus", "in addition", "besides", "although", "a small portion of", "mostly..... but.....", etc. A problem is that few regions are geographically homogeneous. What we want is a description that is applicable to the largest portion of a region, expressing its *essential* characteristics, without having irrelevant information or descriptions that can only be applied to a small part of that region. There is certainly a tension between (1) capturing important characteristics of a region, and (2) excluding information applicable only to a small part of a region. The tradeoff between these two aspects helps to decide what primitive geographical regions are and what informaticn is to be included for each such region.

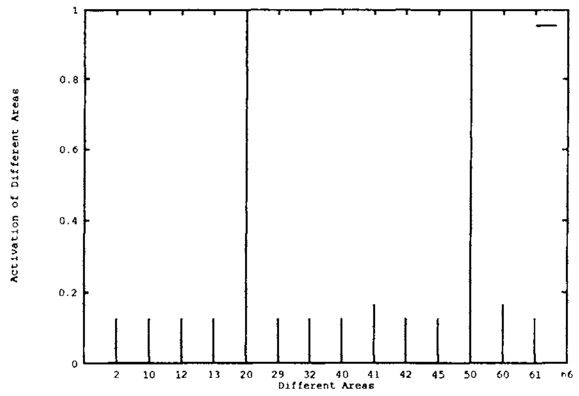
Now we are ready to describe the working of the system. Once a name of a geographical area is given to the system, as imposing a query, the system will find out its agricultural characterization, such as "cattle-country", "rice-growing-area", or "rubber-producing-area", through rule application or similarity matching, or a combination of the two. For example, let us choose to reason about "Brazil-north", which is described as "tropical rainforest hilly plateau". We will start by giving a query: What is the main agricultural product of "Brazil-north"? That amounts to activating the node representing "Brazil-north". To answer this question, we let the system run to perform its reasoning. The output is as follows:

```
>(consyderr 0)

 TITLE:  GEOGRAPHY
 focusing on context AGRICULTURE : remove feature NIL
 setup done
 starting running
 top down
 cl propagating
 cd propagating
 bottom up

 the average activation is 0.1213409896658248
 (2, "cattle-country", 0.1249998807907104)
  (10, "fruit-veg-growing-area", 0.1249998807907104)
  (12, "producing-banana", 0.1249998807907104)
  (13, "producing-tropical-fruits", 0.1249998807907104)
  (20, "rubber-producing-area", 0.9999990463256836)
  (29, "c-Peru", 0.125)
  (32, "Bolivia-orient-rainforest", 0.125)
  (40, "Guiana-pgs", 0.125)
  (41, "Guiana-hilly-country-forest", 0.1666666666666667)
  (42, "Guiana-hilly-country-savanna", 0.125)
  (45, "Brazil-cw", 0.125)
  (50, "Brazil-n", 1)
  (60, "Columbia-basin", 0.1666666666666667)
  (61, "Ecuador-coast", 0.125)
  (66, "Suriname-plateau", 0.125)
```

The result shows that it is a rubber-producing area for sure (with confidence value equal to 0.999999), and it is similar, to a small extent, to "Guiana hilly country" and "Bolivia orient rainforest area" etc. If we want to choose one answer out of many, we can simply use a winner-take-all network on top of this, but this is not an intrinsic part of the system. See Figure 6.

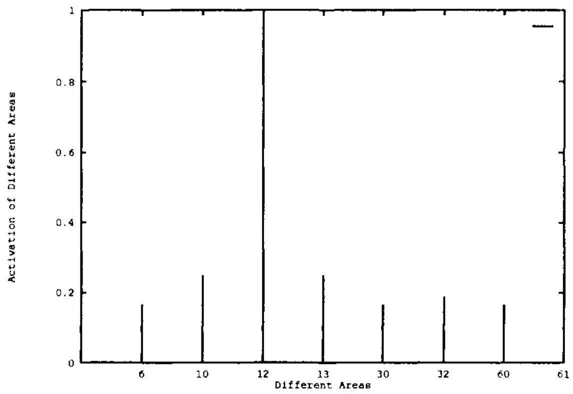


2	"cattle-country"	0.1249998807907104
10	"fruit-veg-growing-area"	0.1249998807907104
12	"producing-banana"	0.1249998807907104
13	"producing-tropical-fruits"	0.1249998807907104
20	"rubber-producing-area"	0.9999990463256836
29	"c-Peru"	0.125
32	"Bolivia-orient-rainforest"	0.125
40	"Guiana-pgs"	0.125
41	"Guiana-hilly-country-forest"	0.1666666666666667
42	"Guiana-hilly-country-savanna"	0.125
45	"Brazil-cw"	0.125
50	"Brazil-n"	1
60	"Columbia-basin"	0.1666666666666667
61	"Ecuador-coast"	0.125
66	"Suriname-plateau"	0.125

Figure 6 Output From the System: Case 1

Another example is as follows: suppose we want to know about the Ecuador coastal area, we will give the system a query: What is the main agricultural product of "Ecuador-coast"? by activating the node representing "Ecuador-coast". To answer this question, we let the system run to perform its reasoning. The output is in Figure 7. The result indicates that the area is producing banana (with confidence value equal to 0.99999) and is very likely producing tropical fruits and other fruits/vegetables. It is similar, in some way, to "Uruguay-coastal", "eastern-Peru" and "Columbia-basin".

As yet another example, let us reason about "Brazil-south". We will start by giving a query: Does "Brazil-south" produce cattle? by activating the node representing "Brazil-south" and looking for "cattle" in the results. The output



6	"Uruguay-coastal"	0.1666666666666667
10	"fruit-veg-growing-area"	0.2499997615814209
12	"producing-banana"	0.9999990463256836
13	"producing-tropical-fruits"	0.2499997615814209
30	"e-Peru",	0.1666666666666667
32	"Bolivia-orient-rainforest",	0.1875
60	"Columbia-basin"	0.1666666666666667
61	"Ecuador-coast"	1

Figure 7 Output From the System: Case 2

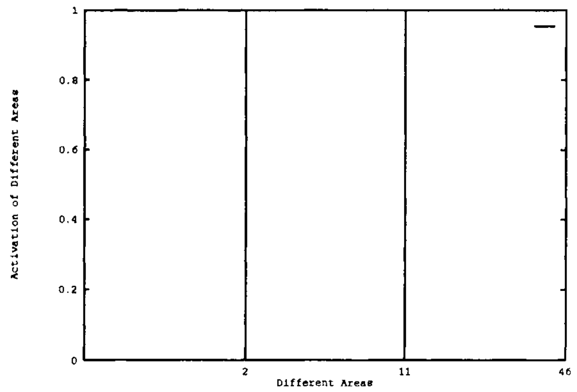
is in Figure 8. The result indicates that the area does produce cattle and sheep. Nothing else in the network fires strongly or distinguishably in this case.

4.2 Other Applications

Now the question is: Can this same method be applied to other domains where no such natural division of concepts and features seems to exist? We will show that the same approach does work for other domains. Due to the space limitation, only some brief hints as to how this architecture can be applied to these other domains will be provided.

Applications to Natural Language Understanding

Natural language understanding is an area in which commonsense reasoning is crucial. For practical purposes, we can either perform a thorough domain analysis to identify useful microfeatures along with concepts, or use some statistical methods to determine similarities and, based on that, construct mi-



2	"cattle-country"	0.9999990463256836
11	"sheep-country"	0.9999990463256836
46	"Brazil-s"	1

Figure 8 Output From the System: Case 3

crofeature representation (see Appendix for details). Microfeatures obtained in this way, unlike in the geography domain, are generally uninterpretable.

One simple example regarding lexical disambiguation (cf. [40] and [2]) and is "Pot" (taken from [18]):

John put the pot inside the dishwasher, because the police are coming.

The point is that normally the word "pot" should be interpreted as "cooking pot", but under certain circumstances, given some pertinent clues, it should be interpreted as marijuana.

A set of situations (which are not provided in [18]) is devised to test a system's ability: (1) John put the pot inside the dishwasher (the solution should be "cooking pot" in this case); (2) John put the pot inside the dishwasher, because the police are coming (the word "pot" means "marijuana" in this case); (3) John put the pot inside the dishwasher, because the police are coming and John wants to make the kitchen clean (the solution should be "cooking pot" in this case); (4) John put the pot inside the dishwasher, because John wants to make the kitchen clean (the solution should be "cooking pot" in this case); (5) John put the pot inside the dishwasher, when the police come for the bankrobbery across the street (the situation is pretty ambiguous, and the system could interpret it

1. Pot is cooking pot.
2. Pot could be marijuana.
3. Marijuana is illegal.
4. Cleaning kitchens implies cleaning cookware.
5. Using dishwashers implies cleaning cookware.
6. If one is having marijuana and the police are coming, then the police will see it.
7. Police seeing illegal substance results someone being arrested.
8. To avoid arrests, prevent the police seeing illegal substances.
9. To prevent somebody seeing something, hide it.
10. Putting something in a dishwasher is for washing it.
11. Putting something in a dishwasher could be for hiding it.
12. If there is a bankrobbery going on and the police are coming, then they are here to stop the bankrobbery.

Figure 9 A List of Rules (weights are omitted).

as "marijuana"); (6) John is cleaning the kitchen, putting the pot inside the dishwasher, when the police come for the bankrobbery across the street (the solution should be "cooking pot" in this case).

A system is constructed based on CONSYDERR, which can solve the original problem and pass the six tests, as follows: Each of the concepts involved in the problem description is represented by one node in the top level. Knowledge in the form of rules is extracted from commonsense knowledge about the concepts involved in the story, as in Figure 9. The bottom level is basically a distributed version of the top level, which allows the sharing of microfeature nodes among the representations of related concepts, so that continuity/similarity can be explored. This structure is constructed using STSIS (see Appendix). The rules are duplicated diffusely in the bottom level.

When the parameters are appropriately set, the system performs the task correctly. It gives correct answers to all tests, distinguishing the often very subtle differences through rule application and similarity matching.[7]

[7]Note that some subtle linguistic elements are not taken into account in the present implementation, for example, "*when* the police are coming" vs. "*because* the police are coming", etc.

Applications to Mundane Reasoning

Mundane reasoning is another area where CONSYDERR is applicable. Mundane reasoning is used to refer to the type of reasoning that we do daily regarding mundane matters, for example, which chair to sit in, when to eat, etc. The goal of such mundane reasoning is to come up rapidly with an interpretation of a situation or to make a quick decision, given the current context. In applying the two level idea, we have to identify all the concepts and rules involved in a particular task. We also have to identify all the microfeatures associated with the concepts.

Let us look into the "Ted" example [5]. Instead of using constraint satisfaction (as in [5]), we perform rule-based reasoning plus similarity matching. The problem can be stated as follows [5]:

> Ted is seen walking along a pier, dressed like a sailor. Ted launched into an excited monolog on the influence of TV programming. It seems reasonable to conclude that Ted is a professional sailor, and that he is interested in television. But another possibility is that Ted is a TV tycoon and a millionaire playboy and has a hobby of sailing.

The point is that normally Ted should be taken to be a professional sailor, but under certain circumstances, given some pertinent clues, Ted should be interpreted as a hobby sailor.

The knowledge used for performing this type of mundane reasoning is encoded in rules (with associated weights); see Figure 10. In the bottom level, distributed representation is used so that similar concepts have shared nodes in that level. It is constructed according to STSIS, and each node is vaguely interpretable.

A set of tests is devised to verify the correctness of the system the same way as before. The system works as expected; for example, when given the input that "Ted is dressed up like a sailor" (in the form of activating nodes "dressed-like-sailor"), the system will indicate that Ted is a sailor (in the form of activating nodes "sailor" strongly); when given the input that Ted is dressed up like a sailor but talks a lot about the TV business (in the form of activating nodes "dressed-like-sailor" and "talk-about-TV-business"), the system will indicate hobby sailors (in the form of activating nodes "hobby-sailor" more strongly).

When finer distinctions are needed, more nodes will have to be added into the system, along with possibly other mechanisms, to take those elements into consideration.

1. dressed-like-sailor talks-like-sailor walks-like-sailor $\longrightarrow$ sailor
2. talk-about-TV-business $\longrightarrow$ interested-in-TV
3. talk-about-TV-business $\longrightarrow$ in-TV-business
4. in-TV-business $\not\longrightarrow$ sailor
5. dressed-as-sailor $\longrightarrow$ hobby-sailor
6. hobby-sailor $\longrightarrow$ rich-people
7. rich-people $\not\longrightarrow$ sailor

Figure 10 A List of Rules (weights are omitted).

Applications to Planning

Planning is yet another area in which CONSYDERR might be applicable (mainly of concern here are commonsensical planning activities, not formal planning based on strict mathematical models). The most important problem is that the sequential nature of the planning domain has to be adequately dealt with. The planning problem is inherently sequential: a plan is formed from a sequence of steps; moreover, steps can interfere with each other in the form of undoing what was accomplished or disabling what should be done, etc. Some notions of temporality have to be incorporated in order to express sequences; and actions, conditions, and results have to be associated with temporal measures.

CONSYDERR, by itself, is non-sequential. So other means need to be employed. This is where the idea of *temporal simulation* comes into play. By temporal simulation, it is meant actually carrying out plan steps temporally inside a system when forming a plan. Since a system can deduce within a system cycle what the next step should be and what the resulting state will be from applying the step, when it is given the current state, then another cycle can further deduce yet another step and the state resulting from that step, with the previously derived state as the current one. This process can go on and on, until reaching some desired state.

Rules and similarities are used readily in planning. Rules are used to encode the relations between the plan step taken within the current context and the new state after the step is performed and the relations between the current state and the next step to take. Similarities are used for matching a situation not precisely specified in the rule sets, for reaching *plausible* conclusions.

5 COMPARISONS WITH OTHER APPROACHES

Comparing the present approach with PDP models [25], we notice both similarities and some differences. In terms of similarities, both approaches utilize networks of simple processing elements, operate in a massively parallel fashion, and are capable of carrying out continuous functions for capturing flexible reasoning in vague domains. Unlike PDP models, this architecture does not rely exclusively on similarities - rules are implemented that can generate precision as well as flexibility [28, 29, 32]. In terms of similarity matching, instead of producing only the closest match among the (stored) training cases (or clusters of them; as in most PDP models), all similar cases can be obtained at once in this architecture, which facilitates comparisons, explanation generations, and other post-processing. Similarities obtained are fully determined by the similarity measure explained earlier, which is unlike most PDP models in which generalization (based on similarity) is unpredicatable. The architecture does not require long training time (as in case of backpropagation networks; cf. [25]), or long settling time (as in case of Boltzmann machine; cf. [5]).

Comparing the present approach with the traditional rule-based approach (e.g., [44] and [13]) in constructing knowledge-based systems, we see some advantage: the two-level architecture is (potentially) capable of performing most of the functionalities of traditional rule-based systems [35], and it can also deal with similarity-based reasoning in an efficient and massively parallel way. Comparing with some variants of rule-based reasoning, such as probabilistic reasoning ([23] and [26]), this approach for encoding rules is computationally simpler, but takes into account cumulative evidentiality (the ability to accumulate evidence) with efficient computation. The weighted-sum computation used can be viewed as a simplification of probabilistic reasoning, under the assumption of independence of evidence (Sun & Waltz [33]).

Comparing this approach with case-based reasoning [24], there is clearly some similarity: both approaches utilize similarities between the current situation and previously known situations to come up with a plausible conclusion. The differences on the other hand are as follows: (1) rules (compound knowledge statements) are the basic coding mechanisms for both concrete and abstract knowledge in CONSYDERR, which allows a simple, uniform representation that encompasses both cases and rules; (2) unlike most case-based systems, similarity matching (as well as rule application) is done here in a massively parallel fashion, and thus is very efficient.

6 SUMMARY

In this chapter, an approach for structuring knowledge has been proposed that might have wide applicability in various vague domains. The idea for the approach came from the analysis of different kinds of flexibilities in reasoning in vague domains. According to this idea, knowledge is divided into two kinds: (1) compound knowledge statements in the form of rules and (2) microfeatures associated with primitive knowledge statements (concepts). Thus, an architecture composed of two levels, the concept level and the microfeature level, has been developed, which allows both rule application and similarity matching. The combination of rule application and similarity matching facilitates both efficient use of knowledge statements explicitly represented in a system and wider coverage in a domain via plausible connections with these statements. Several experiments have been presented that show the possibility of applying this architecture to a wide variety of domains.

APPENDIX: DETERMINING SIMILARITIES AND MICROFEATURE REPRESENTATIONS

STSIS, or *a Statistical Test-Score procedure for determining Intensional Similarity*, provides an alternative way of building microfeature representations. It is a procedure for constructing microfeature representations based only on similarities (which are obtained through empirical means). (For similar approaches, see [41].) This procedure can be applied to automatically develop microfeatures that have no conceptual interpretation, instead of performing a thorough domain analysis to determine conceptually interpretable microfeatures. This is useful because not all domains have a set of microfeatures well analyzed as in geography.

Assume there is a set of concepts $c = \{c_1, c_2, c_3, ..., c_n\}$, and $\bar{c}$ is the vector composed of all these concepts. Matrix M_2 measures the pairwise similarities between elements of the vector $\bar{c}$, that is,

$$M_2 = S(\bar{c} \times \bar{c})$$

where $\times$ denotes outer-product, and S is the similarity matching measure: S([x]) = [S(x)]. (In other words, the similarity of a matrix is a matrix of the similarities of its elements).

For each matrix element,

$$S(a, b) = \frac{\sum_{i=1}^{n} S_i'(a, b)}{n}$$

where $S_i'(a, b)$ is an empirical measure (i.e., a subjective rating) of the similarity between a and b, ranging from 0 to 1. In other words, $S(a, b)$ is obtained from averaging a large number of subjective ratings. We can also calculate the mean squared error:

$$\delta(a, b) = \sqrt{\sum_{i=1}^{n} (S - S_i')^2}$$

Then for higher-order similarities (those involving three or more concepts), we have

$$M_3 = S(\bar{c} \times \bar{c} \times \bar{c})$$

and

$$M_4 = S(\bar{c} \times \bar{c} \times \bar{c} \times \bar{c})$$

and so on. Ideally, we should have $S_{ab} = \frac{|F_a \cap F_b|}{|F_b|}$, $S_{abc} = \frac{|F_a \cap F_b \cap F_c|}{|F_c|}$, and $S_{abcd} = \frac{|F_a \cap F_b \cap F_c \cap F_d|}{|F_d|}$, etc.

One problem with this approach is that there are too many entries to fill in each matrix, especially in higher order ones. One way to deal with this problem is determining which entry will be zero beforehand and thus avoiding computing that entry; for a large number of entries in these matrices will be zero, which can be determined by examining related entries in the lower order ones (if one of the related entries is zero, then the entry is zero). (*Related entries* are defined to be entries that contain a subset of concepts involved in the original entry).

Another problem is when we should stop, because obviously we do not want too many M matrices (we can produce matrices as many as the number of concepts). There is no theoretical result for determining when to stop. However, we can set up some empirical criteria. For example, we can limit the number of matrices to be no more than half the number of the concepts involved.

Once the similarity matching measures are obtained, a pseudo-code description of the algorithm for constructing microfeature representations based on the similarity matching measures is as follows: suppose we have the following matrices: $M_1, M_2, ..., M_m$. Let the total number of nodes in the bottom level be L. Let the number of nodes for c_i be L_i (ideally, $L_i = |F_{c_i}|$).[8] Define S_i to be L_i.

```
Let i=2
Repeat if i $\neq$ m
.For each set of entries (e.g., $M_i$ (a, b) = $S_{ab}$, and $M_i$ (b, a) = $S_{ba}$),
    allocate an appropriate number of nodes shared among those con-
    cepts in the entries, and subtract the same number of nodes from
    each of the node pools established for the related entries of the
    next lower numbered matrix
.i= i+1
```

Here *related entries* mean entries consisting of a set of concepts that is a subset of the original concept, for example, (a b c) for (a b c d). *An appropriate number of nodes* mean the number of nodes proportional to the similarity matching measure in question. For example, suppose $M_i(a, b) = S_{ab}$, and

[8] L_i is determined based on the principle of similarity-based representation: the more general a concept is, the fewer feature nodes there are for representing it.

$M_i(b,a) = S_{ba}$, so the appropriate number of nodes will be $S_{ab} * L_b = S_{ba} * L_a$. According to the definition of similarities above, the equality always holds. It is because $S_{ab} * L_b = \frac{|F_a \cap F_b|}{|F_b|} * L_b = |F_a \cap F_b|$, and $S_{ba} * L_a = \frac{|F_a \cap F_b|}{|F_a|} * L_a = |F_a \cap F_b|$. However, in reality this equality may not hold: because human similarity judgments and measurements are always error-prone, inconsistency is inevitable. Besides, random noises alone are enough to upset the equality. When inconsistency is encountered, we can use the average of the two instead in the formula.

An issue is the subtraction of nodes from the node pools of the related entries in the lower order matrices. Because of the fact that a high-order similarity is part of some lower-order similarities, the (feature) nodes used in the high-order similarity are part of the node pools of the lower-order similarities. When we allocate nodes for a high-order entry, we must subtract the same number of nodes from each of the related lower-order entries. Since we establish node pools for similarities iteratively, from the lowest order up, each time we only need to subtract from the related entries of the closest lower-order matrix.

The question of which nodes to remove from a pool of nodes can be answered partially by considering constraints we have, that is, the fact that we have to preserve established similarities (i.e. node sharing situations). When the constraints we have are not enough to determine a uniquely correct way of removing nodes, we can make a tentative decision and backtrack later if necessary, that is, performing a search over the space of all possible ways of removing nodes, until a test shows that all similarity matching measures are implemented correctly.

REFERENCES

[1] J. Anderson and E. Rosenfeld, (eds.) *Neurocomputing*, MIT Press, Cambridge, MA. 1988

[2] L. Bookman, A Connectionist Scheme For Modeling Context, In D. Touretzky et al. eds. *Proc.1988 Connectionist Summer School*, pp.281-290. San Mateo, CA: Morgan Kaufman, 1989

[3] C. Chang and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Academic Press, San Diago, CA. 1973

[4] A. Collins and R. Michalski, The Logic of Plausible Reasoning: A Core Theory, *Cognitive Science*, Vol 13, No 1, pp.1-49, 1989

[5] M. Derthick, Mundane reasoning by parallel constraint satisfaction, TR CMU-CS-88-182, Carnegie-Mellon University, 1988

[6] D. Dubois and H. Prade, An Introduction to Possibilistic and Fuzzy Logics, in: P. Smets et al (eds.) *Non-Standard Logics for Automated Reasoning*, Academic Press, San Diago, CA. 1988

[7] J. Feldman, Neural Representation of Conceptual Knowledge, Technical Report 189,Department of Computer Science, University of Rochester, 1986

[8] L. Fu, Recognition of Semantically Incorrect Rules, *Proc.IEA/AIE-90*, 1990

[9] S. Gallant, Connectionist Expert Systems, *Communication of ACM*, V.31(2), pp.152-169, 1988

[10] J. Gelfand, D. Handelman and S. Lane, Integrating Knowledge-based Systems and Neural Networks for Robotic Skill Acquisition,*Proc.IJCAI*, pp.193-198, Morgan Kaufman, San Mateo, CA. 1989

[11] S. Grossberg, *The Adaptive Brain*, North-Holland, New York, NY. 1987

[12] P.J. Hayes, In defence of logic, *Proc.5th IJCAI*, pp.559-565, Morgan Kaufman, San Mateo, CA. 1977

[13] F. Hayes-Roth, D.A. Waterman and D.B. Lenat, eds. *Building Expert Systems*, Addison-Wesley, Reading, MA. 1983

[14] J. Hendler, Marker Passing and Microfeature, *Proc.10th IJCAI*, pp.151-154, Morgan Kaufman, San Mateo, CA. 1987

[15] J. Hendler, *Integrating Marker Passing and Problem Solving*, Lawrence Erlbaum Associates, Hillsdale, NJ. 1988.

[16] K. Hornik, M. Stinchcombe and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, Vol.2 pp.359-366. 1989

[17] R. Lacher et al, Backpropagation learning in expert networks, *IEEE Transactions on Neural Networks*. 3, 62-72, 1992.

[18] T. Lange and M. Dyer, Frame selection in a connectionist model, *Proc.11th Cognitive Science Conference*, pp. 706-713, Lawrence Erlbaum Associates, 1989

[19] H. Leonard, *Principle of Reasoning*. Dover, New York, NY. 1967

[20] R. Michalski, Two-tiered concept meaning, inferential matching, and conceptual cohesiveness, in S. Vosniadou & J. Ortony, (eds.) Similarity and Analogical Reasoning, Cambridge University Press, New York, NY. 1989

[21] M. Minsky, *The Society of Mind*, Simon and Schuster, New York, NY. 1985

[22] M. Posner, (ed.) *Foundations of Cognitive Science*, MIT Press, Cambridge, MA. 1989

[23] J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Mateo, CA. 1988

[24] C. Riesbeck and R. Schank, *Inside Case-based Reasoning*, Lawrence Erlbaum Associate, Hillsdale, NJ. 1989

[25] J. McClelland, D. Rumelhart, and the PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructures of Cognition*, MIT Press, Cambridge, MA. 1986

[26] G. Shafer, *A Mathematical Theory of Evidence*, Princeton University Press, Princeton, NJ. 1974

[27] R. Sun, A discrete neural network model for conceptual representation and reasoning, *Proc.11th Cognitive Science Society Conference*, pp.916-923, Erlbaum, Hillsdale, NJ. 1989 a

[28] R. Sun, Designing inference engines based on a discrete neural network model, *Proc.IEA/AIE*, ACM Press, New York, NY. p.1094, 1989 b

[29] R. Sun, Rules and Connectionism, *Proc.INNC-Paris*, p.545, Kluwer, Netherlands. 1990 a

[30] R. Sun, The Discrete Neuronal Models, *Proc.INNC-Paris*, pp.902-907, Kluwer, Netherlands. 1990 b

[31] R. Sun, The Discrete Neuronal Models and the Discrete Neuronal Models, in: B. Soucek ed. *Neural and Intelligent System Integration*, John Wiley & Sons, New York, NY. 1991 a

[32] R. Sun, Chunking and Connectionism, *Neural Network Review*, Vol.4, No.2, pp.76-78, 1991 b

[33] R. Sun and D. Waltz, Neurally Inspired Massively Parallel Model of Rule-Based Reasoning, in: B. Soucek ed. *Neural and Intelligent System Integration*, John Wiley & Sons, New York, NY. 1991

[34] R. Sun, A connectionist model of commonsense reasoning incorporating rules and similarities. *Knowledge Acquisition*, Vol.4, 293-321, 1992

[35] R. Sun, Beyond associative memories, logics and variables in connectionist networks. *Information Sciences*, Vol.70, 1993

[36] D. Touretzky, *The Mathematics of Inheritance.* Morgan Kaufman, San Mateo, CA. 1986

[37] G. Towell, J. Shavlik, and M. Noordewier, Refinement of approximate domain theories by knowledge-based neural networks. *Proc.AAAI-90*, pp.861-866. Morgan Kaufman, San Mateo, CA. 1990

[38] A. Tversky, Features of Similarity, *Psychological Review*, 84(4), pp.327-352, 1977

[39] S. Vosniadou & J. Ortony, (eds.) *Similarity and Analogical Reasoning*, Cambridge University Press, New York, NY. 1989

[40] D. Waltz and J. Pollack, Massively Parallel Parsing, *Cognitive Science*, 1985

[41] L. Zadeh, Fuzzy Sets, *Information and Control*, 8, pp.338-353, 1965

[42] L. Zadeh, Test-score semantics for natural languages and meaning-representation via PRUF, in: B. Rieger, (ed.) *Empirical Semantics.* pp.281-349. Bochum: Brockmeyer. 1981

[43] L. Zadeh, Fuzzy Logic, *Computer*, Vol.21, No.4, pp.83-93, April 1988

[44] M. Waterman, *An Introduction to Expert Systems.* Addison-Wesley, Reading, MA. 1985

9

A Framework for Integrating Relational and Associational Knowledge for Comprehension

LAWRENCE A. BOOKMAN

Sun Microsystems Laboratories
Chelmsford, MA 01824

1 INTRODUCTION

Two important aspects of understanding a text are the ability to skim it, extracting important elements (a coarse-grain view of comprehension), and the ability to read it "deeply" (a fine-grain view of comprehension). A computational analogue that mimics skimming should include a representation of a set of semantic relationships about the text that can be used to summarize it and extract what is important. A computational analogue that supports a deep reading of the text should be able to represent the background details (nonsystematic relationships) associated with the concepts in the text, including the larger frame in which the text concepts are situated.

This chapter describes a two-tier view of semantic memory that supports two complementary views of comprehension mentioned above: a "fine-grain" view that captures the many details of interaction between context and background knowledge as temporal trajectories through "concept space," i.e., the semantic features active in memory at specific points in time. Together these trajectories represent a history of the associational knowledge of the concepts in semantic memory activated by the input, and this activated knowledge contributes to an understanding of the text. A second view, the "coarse-grain" view, captures in the form of a weighted semantic graph, called an interpretation graph, a set of explicit semantic relationships that can be used to reason about the "meaning" of a text.

The semantic memory architecture consists of a relational and an associational tier. The top tier, the relational tier, represents the regularities underlying the structure of our cognitive world expressed as a set of named relationships between concepts. The bottom tier, the associational tier, represents the common or shared knowledge about the concepts in the top tier, expressed as a set of statistical associations. The associational tier encodes the background frame (Fillmore [14]) associated with these concepts, in terms of an underlying substrate of semantic features. Because of the graded character of these semantic features, I will hereafter refer to them as *analog semantic features* or ASFs. The ASFs were developed from the category structure of a thesaurus.

Fillmore [14] argues that our understanding of a concept is determined by how well the conditions of the background situation match the concept's prototype background frame. For example, according to Fillmore, to understand the word *breakfast* "is to understand the practice in our culture of having three meals a day, at more or less conventionally established times of the day, and for one of these meals to be eaten early in the day, after a period of sleep, and for it to consist of a somewhat unique menu." Yet as he points out, each of the above three conditions typically associated with it can be independently absent, still allowing a native user to use the word. For instance, someone can sleep through the morning, wake up at two o'clock in the afternoon, and sit down to a meal of pancakes, bacon, and orange juice, and still call that meal *breakfast*. Thus, the word *breakfast* can be used and understood in a variety of different contexts, as long as the conditions of the background situation more or less match its background frame.

By viewing comprehension as, in part, the activation of temporal patterns through an individual's concept space, we can compare computationally the activated background knowledge (which resides in long-term memory) and activated context (which is activated at the moment of hearing or reading) associated with two different text passages. If their trajectories (as generated by the reader of the passage, e.g., the program) match, these passages will have a similar interpretation, and hence be similarly understood; the degree of similarity depends on the degree of match.[1] Some neurophysiological justification for this view comes from evoked-potential data recorded during reading. For example, it is hypothesized that the sustained information-specific firing patterns that represent primary memory potentials (observed in the frontal lobe) indicate that contextual information is maintained between successive

[1] Many issues of comprehension are beyond the scope of this book. For example, a passage can have more than one interpretation, depending on the reader's intent, goals, and motivation, so there can be several different time trajectories, each reflecting the reader's current state. The reader's perspective can also influence which particulars of a story will be remembered and understood (Anderson and Pichert [3]; Kozminsky [22]). For example, consider a passage about the physical condition of a house: if the reader is a home buyer, the physical condition of the house is important, whereas the contents are probably of little value; if the reader is a burglar, however, the contents are of prime importance (Thorndike and Yekovich [40]).

words in a sentence (Halgren [17]). These temporal firing patterns can be thought of as a set of time-directed trajectories through the concept space of the comprehender of the text.

Comprehension is more than just a passive trace through concept space. Comprehension is also partly driven by our expectations (Schank [32]). Incoming information sets up a context of its own, indicating what is likely to follow. If our expectations are met, we say we comprehend; if not, we may fail to comprehend. Part of this expectation-driven comprehension process is captured by the relational tier of semantic memory in the model. Activation of the relations sets up an expectation of what patterns of knowledge are likely to follow. If incoming information is consistent with these patterns (i.e., if there is semantic overlap), these patterns are reinforced and eventually become part of the interpretation of the text. If not, they simply decay.

As an illustration of these two complementary views of comprehension consider the sentence from Rumelhart [31]: "I was brought into a large white room and my eyes began to blink because the bright light hurt them." When subjects were presented with this sentence and asked what scene comes to mind, Rumelhart found that most people believed that either this was an interrogation situation in which the protagonist is being held prisoner or, it was a hospital scene in which the protagonist is a patient. What information suggests these interpretations? According to Rumelhart, "was brought" is apparently the key, as it evokes a passive situation. Details — the large white room and bright light — further specify the passive situation. A reader attempting to understand the meaning of this sentence would (with the semantic encoding proposed in this chapter) activate the appropriate conceptual relations (e.g., a person is put into a passive situation) based on the reader's and author's shared associational (ASF) knowledge of the concepts "bring into," "large white room," and "bright light."

Figure 1 depicts a relational "coarse-grained" view of the activated concept space in the form of a directed graph. The figure shows the explicit semantic relations that are needed to comprehend the interrogation interpretation of this example. The outlined nodes, labeled "interrogation" and "bring into" represent the *conceptual roots* (i.e., basic events) of the interpretation, from which a baseline summary can be generated: I was brought into a room where I was to be interrogated.

Figure 2 depicts an associational "fine-grain" view of the activated concept space, showing the actual pattern of background knowledge relevant to the given input as a trajectory of semantic features through this space. The figure

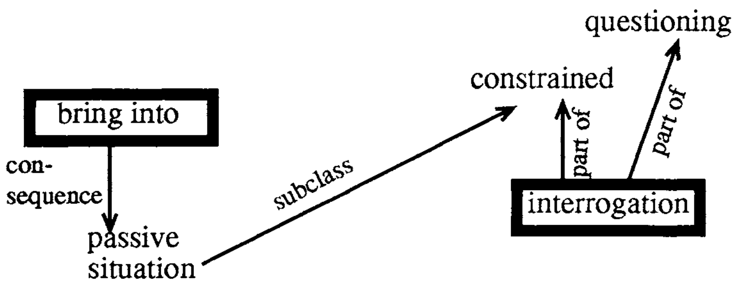


Figure 1 A representative interpretation graph for the sentence "I was brought into a large white room and my eyes began to blink because the bright light hurt them" that supports the interrogation interpretation. The outlined nodes represent the basic events or conceptual roots of the interpretation.

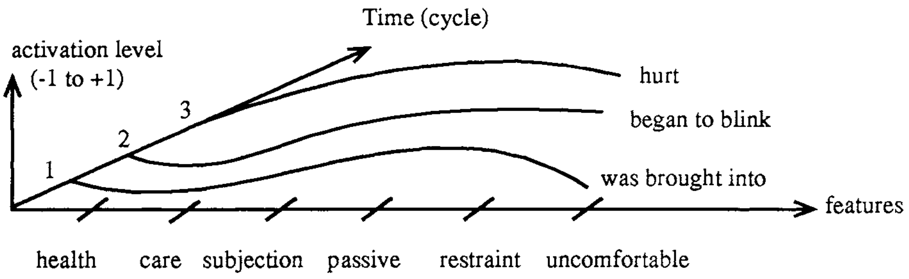


Figure 2 A set of partial time trajectories through feature (ASF) space of the activated background knowledge that supports the interrogation interpretation. The trajectories show what features (ASFs) are active after the processing of each input clause. The clauses containing the concepts "was brought into," "began to blink," and "hurt" were input at cycles 1, 2 and 3, respectively. The figure indicates that the features restraint, passive, uncomfortable, and subjection are strongly associated with this interpretation, and the features health and care are less strongly related to it.

shows some of the details needed to support the interrogation interpretation. For example, after processing the first input clause, "I was brought into a large white room," the features passive, restraint and subjection become activated. (The features health and care are minimally activated.) Processing the second input clause, "my eyes began to blink," the feature uncomfortable becomes activated, while the features passive, restraint and subjection remain activated. After processing of the third clause, "because the bright light hurt them," the features health and care become activated. Thus, interrogations involve restraint and subjection of the person interrogated, and are related to the care and health of that person; furthermore, the interrogated individual is put into a passive situation and usually made uncomfortable.

This chapter argues for an integrated architecture that supports both structured and non-structured representations. Structured representations provide explicit intelligibility and human comprehensibility. Non-structured representations permit similarity-based comparisons between texts that may have some perceived similarity, but which have no explicit connections between them. Both types of representation are needed to support more detailed graded text representations.

The chapter is organized as follows. Section 2 provides an overview of the architecture and the interaction of its components. Section 3 describes the details of the memory architecture. Sections 4 and 5 describe how knowledge is represented in long-term memory. Section 6 analyzes the behavior of the underlying architecture. Section 7 describes the underlying algorithm. Section 8 concludes with a summary.

2 OVERVIEW OF LEMICON

To demonstrate the utility of the two-tier view of semantic memory, a series of ablation experiments was performed on variations of two texts from the stock market domain. The experiments were designed to explore the model's representation of knowledge with regard to the text comprehension task. These experiments are implemented in a computer program named LeMICON (Learning Memory INtegrated CONtext). LeMICON is a structured connectionist model[2] that makes use of both connectionist and symbolic techniques to construct plausible interpretations of text.

[2]The term structured connectionist model refers to a model which represents its knowledge over a set of named nodes. This is similar to the way knowledge is represented in marker-passing semantic networks (e.g., Charniak [8]; Granger et al. [16]; Norvig [28]), except that

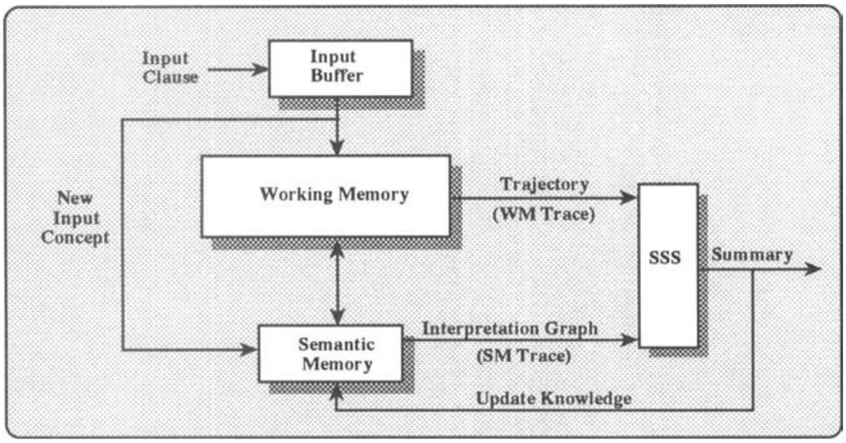


Figure 3 The flow of activation through the system.

Figure 3 depicts each of the model's components along with the flow of activation between these components. Working memory represents a history of LeMICON's associational knowledge. This accumulated knowledge results from the semantic feature activations from semantic memory, and the current input to the system. The links between semantic and working memory represent the bi-directional pathways from which these activations are transmitted.

The input to LeMICON is a text that has been pre-parsed into a set of clauses that are encoded by a set of predefined semantic features extracted from the categories of Roget's thesaurus. This encoding represents the background frame of the clause (see Appendix): the input buffer holds the representation of the next input clause of the text. Each input clause is encoded automatically using co-occurrence statistics on a text corpus (see Bookman [6]). LeMICON produces as one of its outputs an interpretation graph constructed from the active relations (the trace from semantic memory) which reside in the relational tier of semantic memory. (This graph is LeMICON's internal representation of the text.) The interpretation graph is an input to the program SSS, which produces among other things a baseline summary of the text. The other output of LeMICON is a *trajectory* (the trace from working memory) that represents

the nodes in the structured connectionist networks contain simple numeric processing elements and connections between nodes have weights that represent the strength of their connection or relationship. Distributed connectionist models (e.g., Miikkulainen [24]; St. John [37]) represent knowledge as patterns of activation across unnamed nodes.

the history of active background frame details in working memory. This trajectory is also an input to SSS, which SSS uses to compare the similarity of interpretation of different texts.

3 Text Comprehension

This section describes in detail the two components of LeMICON's memory architecture, semantic memory and working memory, and the relationship between them.

3.1 Semantic Memory: The Relational Tier

he relational tier of semantic memory represents the regularities that underlie our cognitive world. This conceptual structure accounts for important aspects of human communication, such as beliefs, preconditions, and knowledge of cause-effect relations (Velardi et al. [41]). There are several ways of expressing these systematic underlying regularities. In semantic network representations they can be expressed as a set of named relationships between concepts (e.g., Alterman [1]; Miller et al. [26]; Norvig [28]; Alterman and Bookman [2]). In such networks the meaning of a concept is represented by its position in the network, that is, in terms of the nexus of relationships that encompass it. However, such networks typically do not indicate the strength of the relationship between concepts. This lack, together with a non-local node computation, makes it difficult for these networks to handle change, or to reinterpret data in the light of new evidence. Instead, they must include a serial evaluation mechanism to select the most relevant interpretation from the generated candidate interpretations. Structured connectionist models (e.g., Lange and Dyer [32]; Shastri [33]; Shastri and Ajjanagadde [34]; Sun [39]; Waltz and Pollack [42]) attempt to remedy this deficiency by attaching weights to the connections between concepts and accumulating evidence local to the node, but they do not yet have any methods for automatically generating these strengths, or the underlying structure. In these networks, concepts are activated based on the amount of evidence available locally, given the current context, to the given node (concept). This obviates the need for a serial evaluation mechanism, since each potential interpretation can be represented by activation in different local areas, and thus can be evaluated in place, and in parallel.

As LeMICON is a structured connectionist model, it is able to handle reinterpretation in the light of new evidence without the need for storing all the potential interpretations it generates. Yet such models assume some pre-existing set of relationships (i.e., some implicit structure) from which to generate these interpretations. This begs the question of how these relationships are learned in the first place. One technique is to construct the network based on co-occurrence statistics gathered from on-line textual corpora. Another constructs a network based on a clustering of a concept's background frame knowledge. (Bookman [7]) describes these techniques in more detail.)

In LeMICON's relational tier, in contrast to some of the symbolic and structured connectionist models discussed above, the meaning of a concept is not determined solely by its position in the network (i.e., physical closeness may indicate semantic closeness, but not necessarily),[3] but also by the closeness of a concept's background frame (its *ASF closeness*: the degree of semantic overlap.)[4] A strong motivating force for having a relational level is that such a structure represents the relational structure of our cognitive world, and so can be used as a basis for reasoning about "understanding."

Figure 4 depicts a portion of the relational tier for the stock market domain automatically constructed from an initial set of 100 concepts. These concepts were chosen based on how representative they were for the given stock market domain.[5] To determine the relationships between these concepts, I applied a modified form of the average conditional mutual information theoretic measure $\bar{I}$ (see Bookman [7]) to all possible concept pairs (i.e., word forms that represent the concept), looking at single paragraphs of text from the *Wall Street Journal* corpus for the co-occurrence of each pair of concepts.

Figure 4 depicts some of the learned relations in graphic form. It shows a portion of semantic memory, focused about the events recession, economic outlook, earnings outlook and market crash. As this graph shows, there are multiple graded relationships between concepts. For example, earnings outlook is affected by and affects the events recession, inflation, poor [economic] outlook, bright [economic] outlook and investing [in the market]. Also recession is affected by inflation, slow economic growth, poor [economic] outlook,

[3] Since weights between concepts are determined by co-occurrence data from actual text, concepts can be physically close in the network but have weak semantic connections, or relatively separated physically but with strong intermediate semantic connections.

[4] The notion of semantic overlap is similar to one presented in Sun[38].

[5] A concept's representativeness for a given domain can be based on its frequency and memorability.

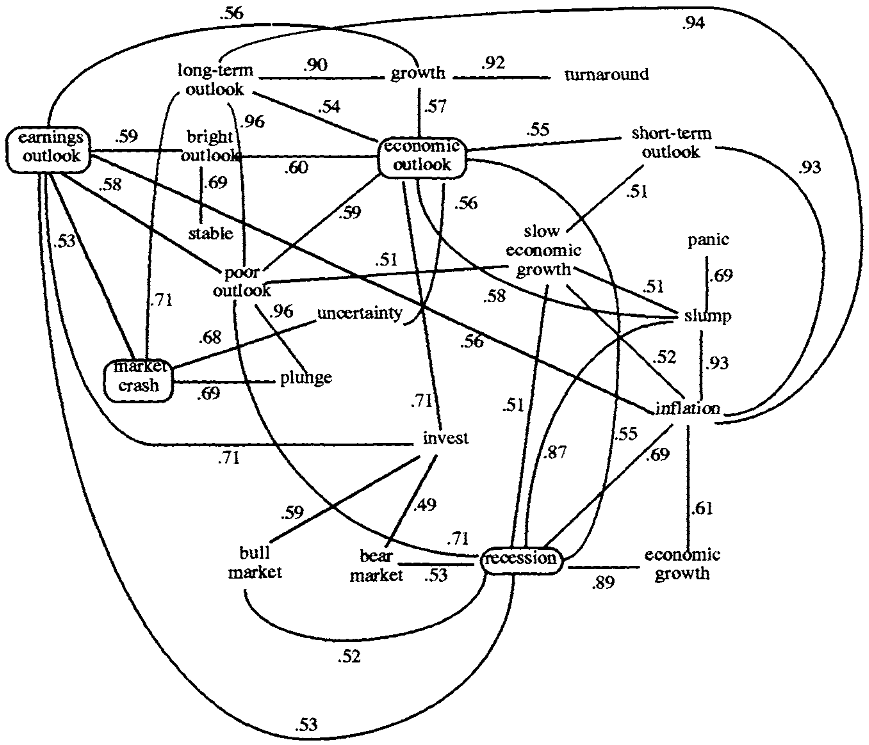


Figure 4 A learned network of relationships derived automatically from co-occurrence statistics gathered from the *Wall Street Journal* corpus. The weight, w, associated with each link represents the strength of the relationship between concept pairs, where $0 \leq w \leq 1$.

and slump [in the market]. It is also affected by both bear market and bull market.

3.2 Semantic Memory: The Associational or ASF Tier

The associational or ASF tier represents the common or shared knowledge about the concepts in the relational tier. This reflects the understanding that each of our concepts has attached to it an associational cloud of knowledge; the tier encodes the nonsystematic knowledge associated with these concepts, what Fillmore [14] calls their background frame, in terms of a set of analog semantic features, called ASFs. The work described here extends the Waltz and Pollack [42] microfeature notion.

How Are the ASFs Chosen?

The ASFs used in this research were chosen on the basis of the category structure of Roget's Thesaurus [4]. The thesaurus is an extremely rich source of knowledge and its structure may provide a clue to how our cognitive world is carved up, i.e., what events, states, and categories we distinguish and use to communicate. As such, it provides an initial approach to encoding detailed knowledge (i.e., background frame knowledge) of how we represent the infinite variety of situations in the world using a finite vocabulary. The use of named features (ASFs) as opposed to unnamed features (such as PDP hidden units, e.g., Hinton, McClelland, and Rumelhart [21]; cf. Hinton [20]) offers a principled way of building in this *a priori* knowledge.

Another use of the thesaurus is described by Morris and Hirst [27], who show how its structure can be used as an aid in determining underlying text structure. For example, they compute what they call the *lexical chains* of a text, i.e., the sequences of semantically related words spanning a topical unit, then show these chains can be used as clues for indicating the intentional structure of the text. Using the thesaurus, Morris and Hirst are able to represent nonsystematic semantic word relationships, which are hard to represent in symbolic frame or semantic network formalisms. For example, it would be difficult to express the relationship between the pair of concepts "interrogation" and "bright light" — a relationship that is essentially nonsystematic — using any fixed set of systematic relationships.

The category structure of Roget's thesaurus can roughly be described ion terms of eight classes: abstract relations, space, physics, matter, sensation, intellect, volition, and affections, with each class further subdivided. The total number of classifications in a thesaurus is extremely large, but 1042 basic classifications form what the thesaurus builders call its backbone structure. For the LeMICON experiments, I chose a subset of 454 of these classifications to represent the set of ASFs. This set is listed in Bookman [7]. Underlying the use of ASFs is the hypothesis that people have idiosyncratic but probably redundant sets of semantic features drawn from their common experiences.

How the ASFs Function

The work here extends the notion of microfeature to include the background frame associated with everyday concepts and their interrelationships. For example, suppose we are given the following list of ASFs: danger, answer, question, opposition, resistance, security, custom, care, discharge, organization, admission, routine, procedure, payment, emergency, and purpose. This list can be used to distinguish the background frame associated with the two concepts, "hospital" and "interrogation." Table 1 shows a partial encoding of these concepts. Notice that the ASFs described there are role independent. The analog property says to what degree these ASFs discriminate these concepts and to what degree the ASFs indicate their similarity. The analog values were generated automatically by computing the mutual information[6] between concept and ASF over a set of sentences from an on-line corpus. Bookman [7] describes an extended technique for encoding ASF knowledge that attempts to deal with the problem of small frequency counts.[7]

Making Fine Discriminations

Consider the following sentences (Waltz [43]):

John nibbled at his food.
John wolfed his food down.

Ideally, we would like to characterize nibble as "to bite off small amounts" and wolf as "devour," not just as instances of "eating." ASFs allow one to make these fine discriminations within a single word sense, in this case the concept "eat." This is another reason for having the ASF level, as these distinctions

[6]The mutual information of two events x and y, $I(x, y)$, is defined according to Fano [11] as follows: $I(x, y) = log_2 \frac{P(x,y)}{P(x)P(y)}$, where P(x,y) is the joint probability of events x and y, and P(x) and P(y) are the respective independent probabilities.

[7]One problem that appeared in the attempt to automatically encode the associational tier using the ASF set was low frequency counts. So, to increase the likelihood of a concept's co-occurrence with an ASF, each ASF was associated with a *dictionary tree* of related words, and the concept then matched against a set of dictionary trees. A dictionary tree encodes an ASF with a set of words that are related in particular ways: synonymy, related to, compared to, contrasted to, and antonymy. A sample dictionary tree for the ASF dislocation shows that dislocation is linked via "synonymy" to the ASFs recession, depression and slump; via the relation "related to" to the ASFs crash, decline, and drop; via "antonymy" to the ASF boom; and via "contrasted to" to the ASFs expansion and growth; it also belongs to the sub-category "relative space" which in turns is a member of the class "space." See Bookman [7] for further details.

Table 1 Parts of the background frames and their association with the concepts "hospital" and "interrogation." The label *I* represents the mutual information value. The higher this value, the stronger the relationship between concept and ASF.

Interrogation	
ASF	Mutual Information (*I*)
resistance	5.4
opposition	5.4
danger	5.3
security	5.3
answer	4.2
question	4.2
customs	4.0
payment	0.0
care	0.0
discharge	0.0

Hospital	
ASF	Mutual Information (*I*)
discharge	4.2
procedure	3.9
emergency	3.6
care	3.5
admission	3.4
routine	3.3
payment	3.1
organization	2.2
purpose	2.2
opposition	1.1

are not readily expressible via logical formalisms such as first-order predicate calculus, unless each semantic variation is explicitly represented.

3.3 WORKING MEMORY

Working memory represents a history of LeMICON's ASF knowledge. This accumulated knowledge results from ASF activations from semantic memory and the current input to the system. Figure 5 shows three different sets of links connecting semantic and working memory. These links represent the bi-directional pathways through which ASF activations are transmitted between semantic and working memory. The solid black links [4] represent the pathways along which activated background knowledge is transmitted, the dark grey links [3] the pathways for each case role pattern, and the light grey links [2] the pathways for ASF relational knowledge between concepts. The case slots for each concept (see Figure 6) represent the other part of the background frame and are filled by the appropriate bank of ASFs in working memory by a process described in Bookman [7].

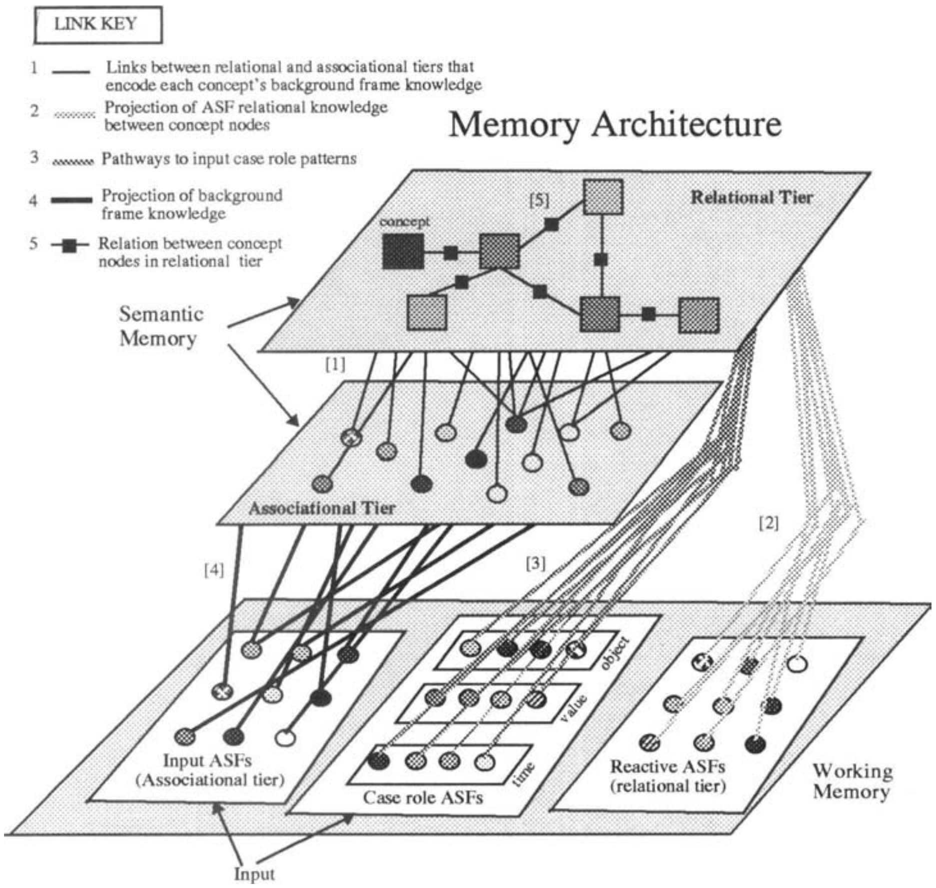

LINK KEY
1 Links between relational and associational tiers that encode each concept's background frame knowledge
2 Projection of ASF relational knowledge between concept nodes
3 Pathways to input case role patterns
4 Projection of background frame knowledge
5 Relation between concept nodes in relational tier
Memory Architecture
Relational Tier
concept
[5]
Semantic Memory
[1]
Associational Tier
[2]
[3]
[4]
object
value
time
Input ASFs (Associational tier)
Case role ASFs
Reactive ASFs (relational tier)
Working Memory
Input

Figure 5 The relationship between semantic and working memory. The solid black links [4] represent the pathways along which activated background knowledge is transmitted, the dark grey links [3] the pathways for each case role pattern, and the light grey links [2] the pathways for ASF relational knowledge between concepts. The case slots for each concept (see Figure 6) represent the other part of the background frame and are filled by the appropriate bank of ASFs in working memory.

Working memory consists of three short-term buffers: (1) the input ASF buffer accumulates ASF background knowledge patterns from both the input text and from the associational tier superimposing the latter patterns onto the former and storing the resulting pattern; (2) the case role ASF buffer stores the ASF case role patterns from the input text (this buffer is flushed after an input clause is processed); and (3) the reactive ASF buffer accumulates the ASF encodings of the activated relations between concepts from the relational tier by superimposing these patterns onto the patterns currently in the buffer.

The buffer that stores the case role patterns is actually segmented into 13 mini-buffers — one buffer for each of the 13 possible case slots. This separation of case slots enables LeMICON to distinguish the semantic roles in the interpretation it constructs. Each mini-buffer holds a unique filler.

4 ENCODING SEMANTIC MEMORY

Symbolically, semantic memory can be thought of as being represented by sets of triples of the form, $(R_{w_{ij}}, C_i, C_j)$, where each triple describes a weighted relationship, $R_{w_{ij}}$, between concepts C_i and C_j of strength w. Attached to each concept (node) are a set of deep cases. What distinguishes the triples from other semantic network formalisms (e.g., see [1, 2, 8, 12, 32, 28, 33, 35, 36]; cf. [39, 42]) is their ASF encoding of associational knowledge and a method for assigning weights between concepts.

Figure 6 provides a more detailed look at a partial ASF representation of one of the triples in the relational tier. This triple (subclass, financial stress, inflation) represents the relation that "inflation" is one kind of "financial stress." The ASF representation of a triple consists of representation of (1) the background frame of each concept, and (2) of the relation between the concepts.

The background frame of each concept is encoded from the knowledge contained in the associational tier of semantic memory automatically, from on-line corpora, via information-theoretic methods as discussed in Section 3.2. This encoding is depicted in Figure 6 by the links connecting the associational tier of semantic memory to each concept in the relational tier of semantic memory. Additionally, each concept has separate slots to hold the potential binding information for each distinct case role filler.[8] For example, Dow Jones fills

[8]Each concept contains 13 case slots, one for each of the 13 possible case relations known to the system.

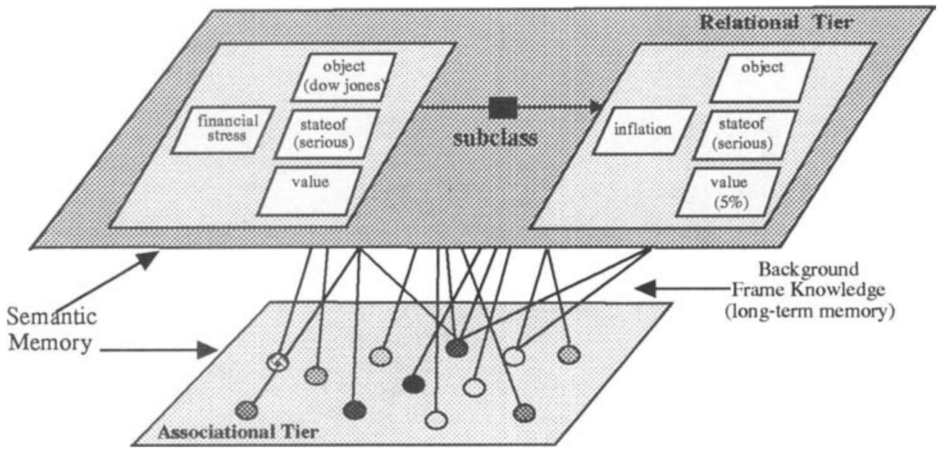


Figure 6 A partial ASF representation of the semantic triple: (subclass, financial stress, inflation). The triple consists of the two concepts "financial stress" and "inflation" linked via a subclass relationship. Each concept consists of two parts: an ASF representation of its background frame, and an ASF representation of its filled case role slots.

the object case slot of financial stress. Each distinct filler is represented by a unique pattern of ASFs linked to the case role slots. Note that these ASFs are kept separate from the ASFs that encode the concept. The ASFs provide constraints on a concept's fillers, in addition to encoding its background frame.

The link connecting the two concepts "financial stress" and "inflation" represents the ASF encoding of the subclass relation between the two concepts. Each link is encoded as $(C_i - C_j)$, i.e., the corresponding difference between the constituent ASF concept encodings. This is intended to reflect any potential change or difference in the relation's "knowledge state."

In LeMICON, the ASFs acting as constraints results in a soft matching capability that allows concepts and the relationships between them to be activated even though not all semantic features match.

5 REPRESENTATION OF SEMANTIC CONSTRAINTS

Each concept in semantic memory contains, in addition to its ASF encoding, a listing of the concept's semantic roles, and any associated antonyms. For example, Figure 7 shows a more detailed representation of the concepts "inflation" and "believe."

CONCEPT	Inflation	Believe
ASF representation:		
Semantic roles:	stateof, object, value	agent, theme, co-theme
Antonyms:	deflation, stability	disbelieve

Figure 7 A description of the semantic information associated with the two concepts "inflation" and "believe." The shaded pattern represents a concept's ASF background knowledge.

The semantic case role slots determine what possible fillers a concept can have. Thus, the concept "inflation" has three semantic case role slots, named *stateof*, *object*, and *value*. Potential fillers of the *stateof* slot might be "serious," as in "serious inflation," or "insignificant," as in "inflation was insignificant." Its *value* slot might include such fillers as "no," as in "no inflation," or "5%," as in "5% inflation." The *antonym* slot is used in conjunction with the mutual information measure described in Bookman [7] to handle the problem of having mutually exclusive information being active simultaneously. A CN-region (*Competitive-Normalized Regions:* Chun, Bookman and Afshartous [10]) is constructed for any relational pair of concepts that is in an antonym relationship.[9]

[9] A CN-region is a network structure that represents the conceptual abstraction of a collection of nodes. The region also provides a computational mechanism for controlling the collective competitive behavior of the nodes. For example, a region can be used to inhibit the activation of such mutually exclusive actions as the rise and fall of the stock market. These structures allow for smooth competitions among concepts in semantic memory, thus enabling subtle differences in meaning to exist. These structures provide more stable competition, are more tolerant to initial noise, and eliminate the premature "lock-in" effect of WTA (Winner-Take-All) structures (Feldman and Ballard [13]).

6 EXPERIMENTS AND RESULTS

In order to test the utility of the semantic memory representation and its underlying architecture, a series of text experiments were undertaken. These experiments took the form of automatically constructing plausible interpretations of a given set of texts.[10] The constructed interpretation takes two forms: (1) a weighted network of relations that represents a "coarse-grain" view of the text; and (2) a set of time-directed trajectories through ASF space that represents the activated background frame knowledge of the comprehender of the text—a "fine-grain" view of the text.

The input to LeMICON is a text that has been pre-parsed into a set of ASF-encoded clauses that represent the background frame of the clause. Each clause is encoded automatically by techniques described in Section 3.2. See Figure 16 for a sample encoding.

6.1 ANALYZING THE OUTPUT AT THE RELATIONAL LEVEL

Consider the texts WSJ-1 and WSJ-2 (see Figures 8 and 9):

WSJ-1: The stock market declined 50 points yesterday. Analysts blamed the slump on the uncertainty in the economic outlook. They believed that further increases in oil prices in conjunction with the current consumer debt level would lead to slow economic growth.

Figure 8 The story WSJ-1.

WSJ-2: The stock market **dropped** 50 points yesterday. **Then investors panicked and the market plunged another 100 points.** Analysts blamed the **drastic change** on the uncertainty in the economic outlook. They believed that further increases in oil prices in conjunction with the current consumer debt level would lead to **market chaos.**

Figure 9 The story WSJ-2. The text highlighted in boldface indicates the differences between this text and the text WSJ-1.

On the surface these texts appear to be similar as they use the same concept vocabulary (e.g., blame, uncertainty, economic outlook, increases in oil prices,

[10]Only two of the texts are analyzed in this chapter. The others are analyzed in Bookman [7].

consumer debt level, and decline/drop appear in both texts). But the two texts are really very different. Although forces affecting the market are the same in both (increases in oil prices and the current consumer debt level), the decline of the market in the text WSJ-2 leads to panic on the part of investors causing a very different situation, namely market chaos, which in concert with the other concepts in the text could lead the reader to infer that the market has crashed. In the text WSJ-1, similar initial conditions (the market declined 50 points in both texts) lead to a very different interpretation, namely, the country may be heading into a recession. The interpretation graph produced from LeMICON's semantic memory can show how to account for some of these differences in the text. One difference between the texts is captured by the summaries produced from these graphs.

Applying SSS [2] to the interpretation graph in Figure 10 yields the following summary:

Baseline summary of WSJ-1: (summary strength = 0.72)[a]

> The stock market plunged 50 points. Analysts say inflation and slow economic growth would lead to a recession and a poor outlook for the stock market.

[a]Summary strength is a confidence measure for the summary—the higher its numeric value, the more confidence we can have in the accuracy of the summary.

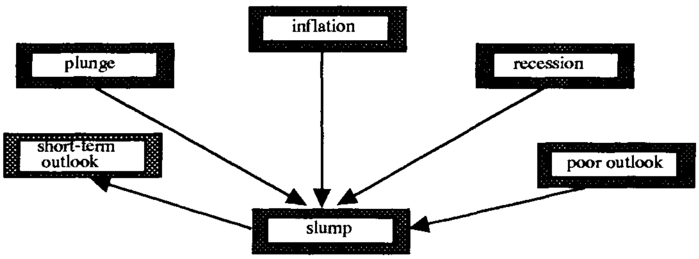


Figure 10 The interpretation graph produced by LeMICON for WSJ-1.

Applying SSS to the interpretation graph in Figure 11 yields the following summary:

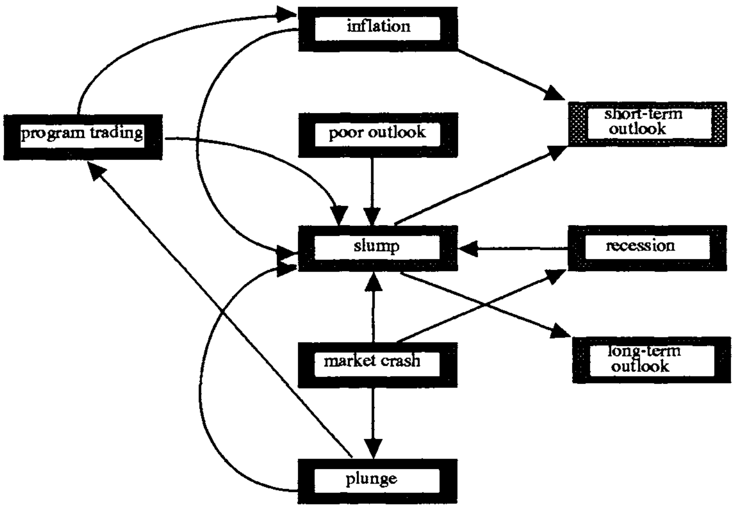


Figure 11 The interpretation graph produced by LeMICON for WSJ-2.

Baseline summary of WSJ-2: (summary strength = 0.64)

The stock market crashed. The outlook for the stock market was poor.

A second difference in the interpretation is reflected in the important events that can be generated from the respective graphs. Importance of a given node in an interpretation graph is defined as the number of nodes reachable from that node in the interpretation graph. In effect, this measure of computing importance is based on the amount of author-emphasized detail. Tables 2 and 3 show the results of applying SSS's importance technique to the interpretation graphs of the texts WSJ-1 and WSJ-2.

Table 2 List of events in order of decreasing importance for the text WSJ-1. An asterisk (*) preceding an event indicates that the event is a conceptual root. Note the average importance is 1.5.

Event	Importance
*plunge	2
*inflation	2
*recession	2
*poor outlook	2
slump	1
short-term outlook	0

Table 3 List of events in order of decreasing importance for the text WSJ-2. An asterisk (*) preceding an event indicates that the event is a conceptual root. Note the average importance is 3.3.

Event	Importance
*market crash	7
plunge	6
program trading	5
inflation	4
recession	3
*poor outlook	3
slump	2
short-term outlook	0
long-term outlook	0

The baseline summaries and important events that LeMICON produces from the interpretation graphs reflect the coarse-level differences found in the texts WSJ-1 and WSJ-2.

6.2 ANALYZING THE OUTPUT AT THE ASF LEVEL

Looking at another aspect of the system's output, the constellations of the ASFs that form in working memory, makes it possible to further analyze the WSJ texts. This dynamic aspect of LeMICON's behavior can be represented as points in *ASF space*. ASF space here refers to an N-dimensional space (for LeMICON N=454), where each element is a vector of length N, and each component of the vector represents an independent ASF. The trajectories in this space refer to the vector of ASFs active in working memory at specific points in time (i.e., after the processing of an input clause). It is hypothesized that the different interpretations and inferences generated (by LeMICON) reflect the readers' associations.

A comparison of LeMICON's behavior as a result of processing the texts, WSJ-1 and WSJ-2, reveals some of these differences. If one plots the change in ASF activity between successive cycles of working memory (a new clause is input on each cycle), a *movie* (i.e., a set of time-directed trajectories) of

LeMICON's behavior develops. As figures 12 and 13 show, the movies for the texts WSJ-1 and WSJ-2 are different.

These differences in understanding are reflected in the activity of the working memory buffer of the two texts that hold the respective ASF background knowledge patterns. The different peaks and valleys displayed in Figures 12 and 13 highlight these fine-grain differences. For example, Figure 12 shows three such differences: (1) region 1 indicates that disappointment and expectation underlie the input sequence of events blame, economic outlook, believe, and slow economic growth; (2) region 2 indicates that a recession underlies the input sequence of events economic outlook, believe, and slow economic outlook; and region 3 indicates that a dislocation underlies the event slow economic growth. Figure 13 also shows three fine-grain differences: (1) region 1 is the same as region 1 in Figure 12, except that disappointment and expectation underlie the input sequence of events plunge, blame, economic outlook, believe, and market chaos (thus, the ASFs disappointment and expectation are common to both texts); (2) region 2 indicates that a depression underlies the input sequence of events blame, economic outlook, believe, and market chaos; and (3) region 3 indicates that these same sequence of events have great breadth, i.e., their scope is broad, as opposed to narrow. Again, the movies for their respective buffers are different.

The above analyses show LeMICON's ability to represent knowledge at a finer level of granularity. This finer granularity permits a deeper understanding of the text because it enables a more detailed analysis of the background details associated with the concepts in the text (see next section).

6.3 A Quantitative Analysis of the Output at the Relational and ASF Levels

How is it computationally possible to determine if two texts are similar in meaning, and thus will be similarly understood? First, a computational measure is defined, called interpretation strength, that compares "weighted semantic graphs." This is a measure of the likelihood of the interpretation, or the strength of the interpretation. Second, another computational measure called *background frame similarity* is defined that compares *interpretation trajectories*, i.e., the generated dynamic "semantic" behavior associated with the comprehension of two texts over time. This latter measure provides a more detailed comparison of the activated background frame knowledge associated with the reader of the text.

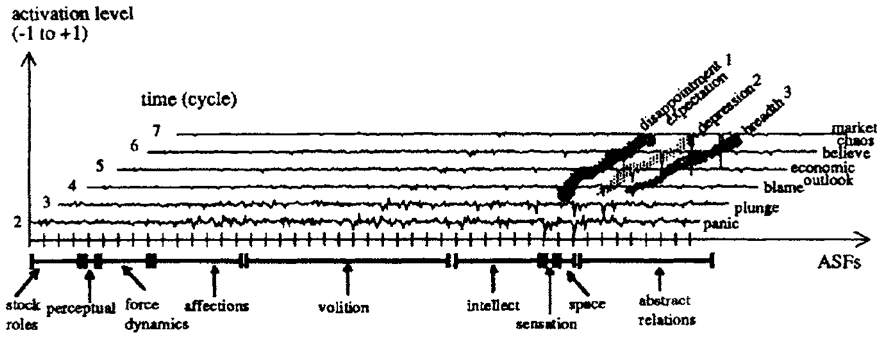


Figure 12 A trace of the change, between successive input cycles, in the activity of the ASFs in working memory for the text WSJ-1. This buffer holds the activated ASF background frame patterns for the text WSJ-1. Note that the clauses containing the concepts blame, economic outlook, believe, and slow economic growth were input at cycles 2-5, respectively. Cycle 1 (decline) was input before the trace begins. The labels along the X-axis represents the ASF range of the classes that roughly reflect the structure of the thesaurus. The shaded regions highlight the key background frame components.

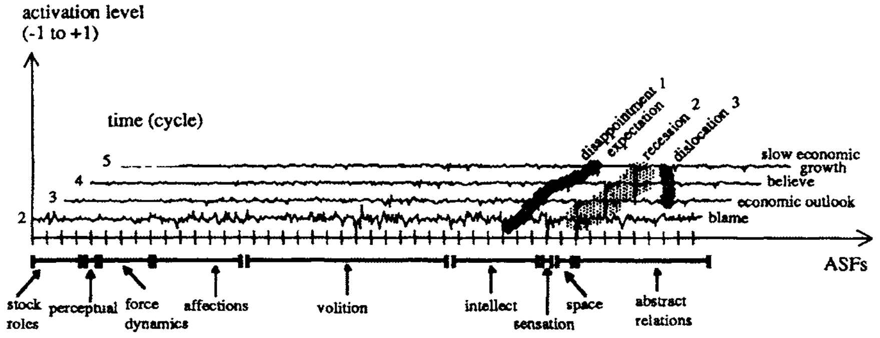


Figure 13 A trace of the change, between successive input cycles, in the activity of the ASFs in working memory for the text WSJ-2. This buffer holds the activated ASF background frame patterns for the text WSJ-2. Note that the clauses containing the concepts panic, plunge, blame, economic outlook, believe, and market chaos were input at cycles 2-7, respectively. Cycle 1 (drop) was input before the trace begins. The labels along the X-axis represents the ASF range of the classes that roughly reflect the structure of the thesaurus. The shaded regions highlight the key background frame components.

Definition 9.1 (Interpretation strength) *The interpretation strength,* IS, *of a weighted semantic graph* G *is*

$$IS(G) = \frac{\sum_{(R_i,a_i,b_i)} \frac{\left(A(R_i)+\frac{A(a_i)+A(b_i)}{2}\right)}{2}}{N}$$

where (R_i, a_i, b_i) is the i^{th} triple in semantic memory, $A(R_i)$ the activation of coherence relation R_i, $A(a_i)$ is the activation of concept a_i. Similarly, for $A(b_i)$. N is the number of triples. The IS(G) *is a value between* $0 \leq IS(G) \leq 1$.

Definition 9.2 (Working memory closeness) *Given the contents of working memory, compute the average ASF closeness of the respective working memory buffers, as measured by the cosine of the angle between the vectors. Given two texts, W_i and W_j, $ASF_{wm}(W_i, W_j)$ denotes their working memory closeness.*

Definition 9.3 (ASF closeness) *The ASF closeness of 2 vectors $\vec{X}, \vec{Y}$ is defined as the cosine of the angle θ between the vectors, where*

$$cos_\theta(\vec{X}, \vec{Y}) = \frac{\vec{X} \cdot \vec{Y}}{|\vec{X}||\vec{Y}|}$$

and

$$|\vec{X}| = \sqrt[2]{x_1{}^2 + x_2{}^2 + \ldots + x_n{}^2}$$

Intuitively, the contents of working memory represent a "compressed" set of trajectories through ASF space, since working memory represents the accumulation of the active ASF patterns (i.e., associations) over time. If the trajectories generated for the two texts match, the hypothesis is, the two texts will have a similar background frame interpretation, and hence be similarly understood, with the degree of similarity determined by the degree of match. Evidence from Heit et al. [18] suggests individual neurons reveal different levels of activation to different events. This supports my hypothesis, since it suggests that a part of the brain actually contributes specific information to the encoding and subsequent recognition of some stimulus during recent memory.[11]

[11] In one experiment, the responses of hippocampal neurons to a particular word or face were recorded, and approximately three-quarters of the neurons tested showed visible and statistically significant evidence for specific activation to one of 10 repeating words (Heit et al. [18]).

Conjecture (Background frame similarity): Texts whose working memory closeness is within some δ are similar in meaning and generate similar sets of inferences, and hence will be similarly understood.

Table 4 Some comparisons of background frame similarity of WSJ texts. WSJ-3 is the text "It was reported that there was financial chaos in the market."

Background Frame Similarity		
Text Comparison	Working Memory Closeness $[ASF_{wm}(x,y)]^{\circ}$	% Change[a] $[\Delta ASF_{wm}(x,y)]$
(WSJ-1,WSJ-3)	50	28
(WSJ-2,WSJ-3)	43	24
(WSJ-1,WSJ-2)	13	7

[a]Percent change calculated against a horizontal baseline of 180°.

Table 4 compares the background frame similarities of the texts WSJ-1, WSJ-2, and WSJ-3.[12] Several observations can be made:

1. In terms of background frame similarity, the two texts, WSJ-1 and WSJ-2, differ in meaning ($\Delta IS = 9\%$ [0.04/0.43] and $\Delta ASF_{wm} = 7\%$ for this example). Small changes in background frame similarity can lead to very different interpretations of the text.

2. The interpretations of the texts WSJ-1 and WSJ-2 ($\Delta IS = 9\%$ and $\Delta ASF_{wm} = 7\%$) are closer in meaning than the interpretations for WSJ-1 and WSJ-3 ($\Delta IS = 30\%$ and $\Delta ASF_{wm} = 28\%$) and the interpretations for WSJ-2 and WSJ-3 ($\Delta IS = 18\%$ and $\Delta ASF_{wm} = 24\%$). Again, this conclusion is in accord with our comprehension of these texts, i.e., it is the strength of the market decline, along with some market forces, "increase in oil prices" and "consumer debt level" which account for this finding. These associations override the fact that the text WSJ-3 is related to both WSJ-1 and WSJ-2 (i.e., "financial chaos" is a part of "recessions" and "market crashes").

Table 6 compares the background frame similarity of the sentence pairs discussed in Bookman [5]. These sentence pairs are reproduced in Table 5. Again, several observations can be made:

[12]An alternate way of comparing interpretations is to compute the background frame similarity over the entire set of generated trajectories.

Table 5 Pairs of ambiguous sentences.

Text Label	Sentence pair
Party	John went to Mary's party. He had a good time.
Rev-party	He had a good time. John went to Mary's party.
Race	John ran the 500 meters yesterday. He had a good time.
Talk	John was talking to his boss. The language he used was inappropriate.
Prog	John was programming at his computer. The language he used was inappropriate.

1. Since the texts Party and Rev-party are in essence the same, i.e., very similar in meaning, they should be similarly understood by LeMICON. A comparison of the trajectories generated for these sentences indicates this to be the case, since the difference between the trajectories as measured by the change in background frame similarity is only 2%.

2. Although the texts Party and Race have the same second sentence, these sentences mean different things. Again, the change in background frame similarity (45%) indicates that a different interpretation was constructed for each text, that corresponded to their differences in meaning. Similar remarks apply to the texts Talk and Prog.

3. According to the background frame similarity measure the texts Prog and Race are the most different in meaning (54%).

Table 6 Some comparisons of background frame similarity for the sentence pairs in Table 5.

Background Frame Similarity		
Text Comparison	Working Memory Closeness $[ASF_{wm}(\text{x,y})]^{\circ}$	% Change $[\Delta ASF_{wm}(\text{x,y})]$
(prog,race)	98	54
(prog,party)	85	47
(party,race)	81	45
(prog,talk)	63	35
(party,rev-party)	6	2

7 ALGORITHM

A functional description of the basic algorithm is shown in Figure 14. The input is a parsed ASF-encoded clause in case notation format. In Bookman [7] a mathematical description and formal analysis of the algorithm is presented. The algorithm has two basic functional components: (1) activation of semantic memory, and (2) update of working memory, i.e., updating the context of the active concepts and relations in semantic memory. Component 1 produces an interpretation graph as output. Component 2 produces a trajectory which represents a history of the active details of the background frames of those concepts in semantic memory that have been activated by the current context.

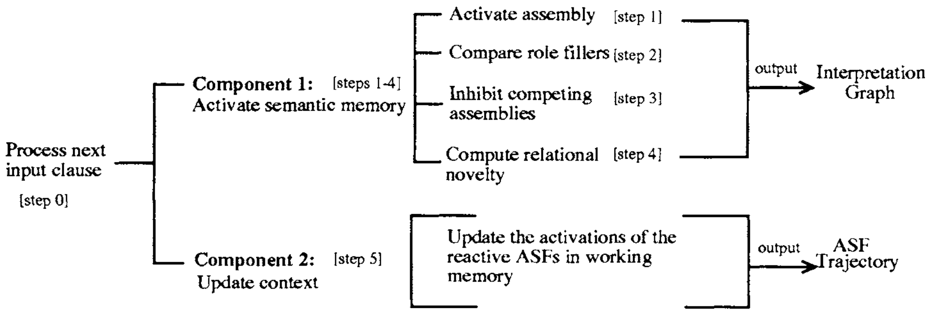

Process next input clause
[step 0]
Component 1: [steps 1-4]
Activate semantic memory
Component 2: [step 5]
Update context
Activate assembly [step 1]
Compare role fillers [step 2]
Inhibit competing assemblies [step 3]
Compute relational novelty [step 4]
Update the activations of the reactive ASFs in working memory
output
Interpretation Graph
output
ASF Trajectory

Figure 14 A functional description of the basic algorithm.

7.1 ACTIVATION OF SEMANTIC MEMORY

As knowledge is represented in LeMICON at both the associational (ASF) level and at the relational level, so two allied notions underlie the activation of concepts in semantic memory — *ASF closeness* and *relational closeness*. Component 1 of the basic algorithm activates the concepts in semantic memory based upon each concept's ASF closeness to the ASFs in working memory and relational closeness to the given input. These two measures of closeness are used in steps 1 (activate concept assembly) and 4 (compute relational novelty) of the computation.

Intuitively, ASF closeness refers to the "semantic overlap" between two sources of information in ASF space. For example, one source of concept activation depends on how close the ASF representation of a concept's background frame is to the input ASF and case role ASF patterns in working memory (step 1). Another source of closeness depends on the ASF encoding of the relation between two concepts and the reactive ASF patterns in working memory (step 4).

Relational closeness refers to the "relational distance" between two concepts in semantic memory, as measured by their propagation strengths, i.e., the time it takes to propagate a signal between two active concepts. Intuitively, concepts with less relational distance are semantically related and will have greater propagation strengths. Here propagation strength is proportional to the average conditional mutual information value (see Figure 4) — the larger the value, the stronger the connection and hence the faster the propagation of information. There are several ways to compute relational closeness. One way is to assume a non-uniform weight between each concept pair (Definition 9.5).

Definition 9.4 (Immediate propagation strength) *The immediate propagation strength between two directly linked concepts X, Y is proportional to the average conditional mutual information, $\overline{I}(X,Y)$.* $\overline{I}(X,Y)$ represents the strength of the relationship between the two concepts, conditional on the concepts X and Y both not occurring together.[13]

Definition 9.5 (Relational closeness) *The relational closeness between two concepts X, Y is defined to be*

$$C_{r_1}(X,Y) = max \prod_{\text{all paths}} \text{(immediate propagation strengths along some path connecting } X \text{ and } Y\text{)}$$

That is, the maximum over all paths, of the products of the immediate propagation strengths along some path connecting the two concepts. This definition assumes that the immediate propagation strength is a real number between zero and one.

For example, given the graph shown in Figure 15, the relational closeness between nodes A and D is 0.54.[14]

[13] See Bookman [7] for a formal definition of $\overline{I}$ and a method of calculating its value.

[14] This value can be calculated by first computing the set of propagation strengths for all paths connecting the nodes A and D.

1. the path ABCD = $(.8 * .8 * .6) = 0.38$
2. the path ACD = $(.9 * .6) = 0.54$
3. the path AD = 0.4

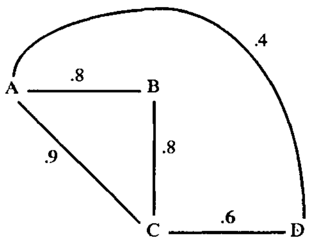


Figure 15 A weighted graph.

7.2 Update Working Memory

Component 2 of the basic algorithm determines how much "knowledge" each concept and relation in semantic memory actually contributes to the ASF patterns in working memory. This is the equivalent of the "credit assignment problem" as applied to text comprehension. In this case the underlying background knowledge is responsible for determining which semantic features are relevant in the current context. As a result of this computation, the input and reactive ASFs in working memory are updated based on the activated ASF patterns in semantic memory, that is, the ASF encodings of the background knowledge of its active concepts and relations.

7.3 Distinctive Features of the Algorithm

There are six distinctive features of the algorithm. These are listed below along with a discussion of why they are important.

Local computation: semantic memory, the algorithm can be efficiently performed by massively parallel hardware, ignoring communication and signal propagation. The experiments in this chapter were computed on a

Note that in this case the shortest path, AD, is not the path that has the smallest propagation strength. An alternative way of computing relational closeness is to assume a unit uniform weight between each concept pair. If one assumes a unit uniform weight between relations in semantic memory, then relational closeness can be implemented using the all pairs shortest path algorithm which addresses the problem of determining a matrix A such that A(i,j) is the length of a shortest path from i to j. A more complicated scheme might take into account the type of the relation and/or the strength of connection between relations, in which case the weight would be non-uniform. For example, Definition 9.5 takes into account the strength of the relation, but not its type.

Symbolics workstation using a general purpose massively parallel simulator called AINET-2 (Chun [9]). In addition, the computation is done without spreading activation over (or relaxing) the nodes and links, unlike the spreading activation/marker passing algorithms described in the literature (e.g., Charniak [8]; Hendler [19]; Norvig [28]).[15]

Activation of relevant knowledge: The ability to access "relevant" knowledge is due to the fact that LeMICON computes a dynamic context based on whatever knowledge is active in semantic memory (i.e., the active relations and activated background knowledge) and whatever knowledge is "contextually active" in working memory. Working memory is a rich source of knowledge — it contains not only the currently relevant (active) case role bindings, but also the relevant bundles of ASFs common to LeMICON's current interpretation of the text; and since the level of activation in working memory can vary smoothly, LeMICON can construct graded interpretations and graded plausible inferences, shifting as the context shifts, without an explicit central interpreter, as prior interpretations that receive little supporting evidence simply become inactive.

Case role information plays an important part in the activation of a concept's background frame knowledge. As shown with the Rumelhart example the details, supplied by the case roles, help further specify the given situation: bright light signaled to the reader that the sentence probably referred to an interrogation. In LeMICON, this effect is achieved by taking the cross product of an input concept's background frame knowledge and the background frame knowledge of its associated case roles with each of the background frames of the concepts in semantic memory (step 1: activate assemblies).

Computing what's new: As discussed in Section 1, our understanding is also driven by our expectations. Part of this process is captured by the relational tier representation of semantic memory as the activation of relations in that tier sets up an expectation about which patterns of knowledge are likely to follow (step 4: compute relational novelty). If incoming information is consistent with these patterns, the patterns are reinforced and eventually become part of the interpretation of the text (i.e., the interpretation graph); if not, they simply decay. The computation in step 4 emphasizes what is "new" about the effect of the current input on the relations in semantic memory, by measuring the difference in activation between previously active and currently active triples. If this difference is small, very little changes. This computation is consistent with recent neurophysiological

[15]Gallant [15] describes a model of word sense disambiguation that also does not involve spreading activation over many cycles.

evidence that familiar, expected, or recently experienced stimuli cause less activation than novel stimuli in parts of the brain (Miller et al. [25]).[16]

Fluid decision making: One additional property of the ASF substrate is its uniform representation of knowledge, which means all ASF information is processed alike. Again, no central executive determines which things might change; rather the temporal association of ASFs in working memory causes the formation of an interpretation or the generation of a set of inferences. As these temporal associations change because of new input, so does the interpretation. Furthermore, the process describes a kind of unsupervised learning: by varying its ASF inputs, the system can dynamically and automatically change the semantics for activating the appropriate concepts and relations in semantic memory (see Bookman [7] for a detailed discussion of this matter).

Hysteresis effects: The system exhibits hysteresis, i.e., the state of LeMICON depends on its previous history. For example,the constellations of the reactive and input ASFs in working memory determines which interpretation will get constructed. Even if LeMICON were to process the same input (or sequence of inputs) for two given texts, it would produce different ASF trajectories if the ASF patterns that precede the input differ. In addition, LeMICON exhibits hysteresis effects related to learning. Bookman [6] reports on the results of a series of five experiments that illustrate these effects. These experiments indicate that prior experience affects what the system learns.

Quick access to knowledge: Quick access to knowledge is also important. As Feldman and Ballard [13] point out, humans carry out complex behaviors in only a few hundred milliseconds. In LeMICON, the activation of relevant knowledge is determined by computing the "ASF closeness" between two concepts or relations. The heart of this computation is rather simple, as it is based on computing inner products, and since all computation is local, it can be done in parallel.

[16]Experiments with monkeys indicate that the formation of memories may in part consist of the modification of synaptic weights such that familiar, expected, or recently seen stimuli cause the least activation in the inferior temporal cortex (Miller et al. [25]). The authors suggest that the inferior temporal neurons may be acting as "adaptive mnemonic filters that seek to preferentially pass information about new, unexpected, or not recently seen stimuli." They state further that such a process is a critical component of the memory storage mechanism. Although these findings were based on experiments with monkeys, the authors state that they are consistent with recent findings in humans. For example, the responses of some face-selective cells in the superior temporal sulcus decline when faces are presented repeatedly (Rolls et al. [30]).

8 SUMMARY

A weakness of traditional network relational representations of semantic memory (e.g., Quillian [29]; Simmons [35]; Fahlman [12]; Sowa [36]) is their limited ability to readily represent human or computer background frames of knowledge. As a consequence, the systems that use such representations to process knowledge are somewhat limited — they cannot readily process the detailed knowledge that underlies each of our concepts, and therefore they cannot easily interpret a text at a deep level of understanding. The architecture described in this chapter, and its associated processing mechanisms represent a very different view of semantic memory: memory is dynamic, it exhibits hysteresis effects, and it represents knowledge at a finer level of granularity. The key components of this architecture are (1) its representation of a concept's background frame knowledge in an associational tier (long-term memory), and (2) its representation of dynamic context via a set of *reactive ASFs* in working memory.

Taken together, these two views espouse a rather novel model of comprehension. In the "trajectory" view, comprehension can be viewed as a dynamic system which changes course (and changes state) by moving to a different point in its associational (ASF) knowledge space as the input changes. At particular points in this space (e.g., after a clause is processed), an interpretation graph can be extracted and used to explain some of the basic properties of the current state, or an ASF trajectory can be generated and used to describe the background details active at specific points in time, thus providing a deeper level of knowledge about the comprehension of the text at these points.

APPENDIX A: SAMPLE PARSED INPUT TO LEMICON

For each story below, a symbolic description of a sample of the parsed input to LeMICON is given. The actual input consists of this parsed input, but with each concept and its filled case slots replaced by their respective learned ASF encodings. For an example, see Figure 16. Each such encoding is a vector of dimension 454.

WSJ-1: The stock market declined 50 points yesterday. Analysts blamed the slump on the uncertainty in the economic outlook. They believed that further increases in oil prices in conjunction with the current consumer debt level would lead to slow economic growth.

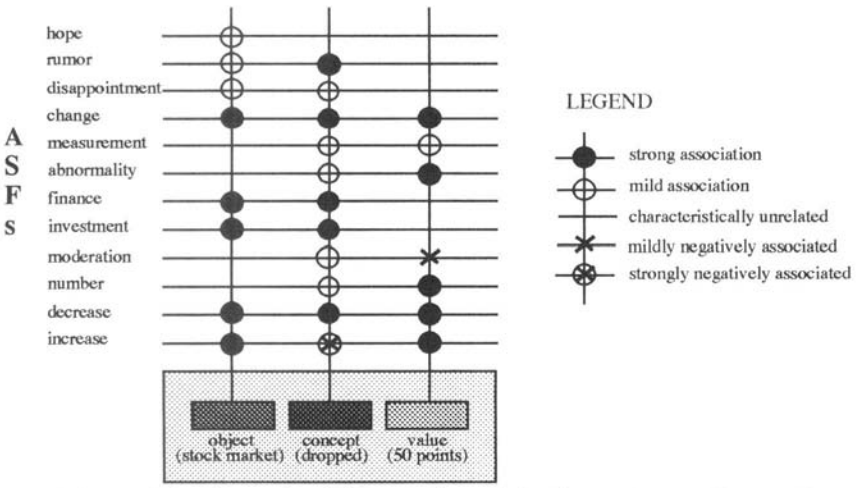


Figure 16 A sample ASF encoding of the input clause "The stock market declined 50 points."

```
Clause 1: (decline (OBJ stock market)
                   (VALUE 50 points)
                   (TIME yesterday))
Clause 2: (blame (AGT analysts) (THM slump))
Clause 3: (economic outlook (STATEOF uncertainty))
Clause 4: (believe (AGT they)
                   (THM increases in oil prices)
                   (COTHM consumer debt level))
Clause 5: (lead (THM slow economic growth))
```

WSJ-2: The stock market dropped 50 points yesterday. Investors panicked and the market plunged another 100 points. Then investors panicked and the market plunged another 100 points. Analysts blamed the drastic change on the uncertainty in the economic outlook. They believed that further increases in oil prices in conjunction with the current consumer debt level would lead to market chaos.

```
Clause 1: (drop (OBJ stock market)
                (VALUE 50 points)
                (TIME yesterday))
```

```
Clause 2:  (panic (AGT investors))
Clause 3:  (plunge (OBJ market)
                   (VALUE another 100 points))
Clause 4:  (blame (AGT analysts) (THM drastic change))
Clause 5:  (economic outlook (STATEOF uncertainty))
Clause 6:  (believe (AGT they)
                    (THM increases in oil prices)
                    (COTHM consumer debt level))
Clause 7:  (lead (THM market chaos))
```

ACKNOWLEDGEMENTS

I would like to thank Ellen Hays for her comments on an earlier draft.

REFERENCES

[1] Alterman, R. (1985). A dictionary based on concept coherence. *Artificial Intelligence*, 25:153–186.

[2] Alterman, R. and Bookman, L.A. (1992). Reasoning abut the semantic memory encoding of the connectivity of events. *Cognitive Science*, 16(2):205-232.

[3] Anderson, R.C. and Pichert, J.W. (1978). Recall of previously unrecallable information following a shift in perspective. *Journal of Verbal Learning and Verbal Behavior*, 17:1–12.

[4] Berrey, L.V. (Ed.) (1962). *Roget's International Thesaurus (3rd edition)*. NY: Thomas Crowell Company.

[5] Bookman, L.A. (1989). A connectionist scheme for modelling context. In *Proceedings of the 1988 Connectionist Models Summer School*, pp. 281–290. San Mateo, CA: Morgan Kaufmann.

[6] Bookman, L.A. (1993). A scalable architecture for integrating associative and semantic memory. *Connection Science*, 5(3&4):243-273.

[7] Bookman, L.A. (1994). *Trajectories through Knowledge Space: A Dynamic Framework for Maching Comprehension*. Norwell, MA: Kluwer.

[8] Charniak, E. (1986). A neat theory of marker passing. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 584–588, San Mateo, CA: Morgan Kaufmann.

[9] Chun, H. (1986). AINET-2 user's manual. Technical Report CS-86-126, Computer Science Department, Brandeis University, Waltham, MA.

[10] Chun, H.W., Bookman, L.A. and Afshartous, N. (1987). Network regions: Alternatives to the winner-take-all structure. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence*, pp. 380–387.

[11] Fano, R.M. (1961). *Transmission of Information.* Cambridge, MA: MIT Press.

[12] Fahlman, S. (1979). *NETL: A System for Representing and Using Real-World Knowledge.* Cambridge, MA: MIT Press.

[13] Feldman, J.A. and Ballard, D.H. (1982). Connectionist models and their properties. *Cognitive Science*, 6:205–254.

[14] Fillmore, C.J. (1982). Frame semantics. In The Linguistic Society of Korea (Ed.), *Linguistics in the Morning Calm.* Seoul: Hanshin Publishing Company.

[15] Gallant, S.I. (1991). A practical approach for representing context and for performing word sense disambiguation using neural networks. *Neural Computation*, 3(3):293–309.

[16] Granger, R.H., Eiselt, K.P. and Holbrook, J.K. (1986). Parsing with parallelism: A spreading activation model of inferencing processing during text understanding. In J. Kolodner and C. Riesbeck (Eds.), *Experience, Memory, and Reasoning*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[17] Halgren, E. (1990). Insights from evoked potentials into the neuropsychological mechanisms of reading. In A.B. Scheibel and A.F. Wechsler (Eds.), *Neurobiology of Higher Cognitive Function.* NY: Guilford.

[18] Heit, G., Smith, M.E. and Halgren, E. (1988). Neural encoding of individual words and faces in the human hippocampus and amygdala. *Nature*, 333:773–775.

[19] Hendler, J. (1986). *Integrating Marker-Passing and Problem-Solving: A Spreading Activation Approach to Improved Choice in Planning.* PhD thesis, Brown University, Providence, RI.

[20] Hinton, G.E. (1981). Implementing semantic networks in parallel hardware. In G.E. Hinton and J.A. Anderson (Eds.), *Parallel Models of Associative Memory*. Hillsdale, NJ: Lawrence Erlbaum Associates.

[21] Hinton, G.E., McClelland, J.L. and Rumelhart, D.E. (1986). Distributed representations. In J.L. McClelland and D.E. Rumelhart (Eds.), *Parallel Distributed Processing: Explorations in the Microstructures of Cognition (Vol. 1)*. Cambridge, MA: MIT Press.

[22] Kozminsky, E. (1977). Altering comprehension: The effect of biasing titles on comprehension. *Memory and Cognition*, 5:482–490.

[23] Lange, T.E. and Dyer, M.G. (1989). High-level inferencing in a connectionist network. *Connection Science*, 1(2):181–217.

[24] Miikkulainen, R. (1993). *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon, and Memory*. Cambridge, MA: MIT Press.

[25] Miller, E.K., Li, L. and Desimone, R. (1991). A neural mechanism for working and recognition memory in inferior temporal cortex. *Science*, 254:1377–1370.

[26] Miller, G.A, Beckwith, R. Fellbaum, C. Gross, D. and Miller, K. (1990). Five papers on WordNet. CSL Report 43. Cognitive Science Laboratory, Princeton University.

[27] Morris, J. and Hirst, G. (1991). Lexical cohesion computed by thesaural relations as an indicator of the structure of the text. *Computational Linguistics*, 17(1):21–48.

[28] Norvig, P. (1989). Marker passing as a weak method for text inferencing. *Cognitive Science*, 13(4):569–620.

[29] Quillian, M.R. (1968). Semantic memory. In M. Minsky (Ed.), *Semantic Information Processing*. Cambridge, MA: MIT Press.

[30] Rolls, E.T., Baylis, G.C., Hasselmo, M.E. and Nalwa, V. (1989). The effect of learning on the face selective responses of neurons in the superior temporal sulcus of the monkey. *Experimental Brain Research*, 76(1):153–164.

[31] Rumelhart, D.E. (1981). Understanding understanding. In H.W. Dechert and M. Raupach (Eds.), *Psycholinguistic Models of Production*, Norwood, NJ: Ablex Publishing.

[32] Schank, R. (1982). *Dynamic Memory.* NY: Cambridge University Press.

[33] Shastri, L. (1988). A connectionist approach to knowledge representation and limited inference. *Cognitive Science*, 12(3):331-392.

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

[35] Simmons, R.F. (1973). Semantic networks: Their computation and use for understanding english sentences. In R.C. Schank and K.M. Colby (Eds.), *Computer Models of Thought and Language.* San Francisco, CA: W.H. Freeman and Company.

[36] Sowa, J.F. (1984). *Conceptual Structures: Information Processing in Minds and Machines.* Reading, MA: Addison-Wesley.

[37] St. John, M.F. (1992). The story gestalt: A model of knowledge-intensive processes in text comprehension. *Cognitive Science*, 16:271-306.

[38] Sun, R. (1991). Connectionist models of rule-based reasoning. In *Proceedings of the Thirteenth Annual Cognitive Science Conference*, pp. 437-442.

[39] Sun, R. (1994). *Integrating Rules and Connectionism for Robust Commonsense Reasoning.* New York: John Wiley.

[40] Thorndike, P.W. and Yekovich, F.R. (1980). A critique of schema-based theories of human story memory. *Poetics*, 9:23-49.

[41] Velardi, P., Pazienza, M.T., and Fasolo, M. (1991). How to encode semantic knowledge: A method for meaning representation and computer-aided acquisition. *Computational Linguistics*, 17:153-170.

[42] Waltz, D.L. and Pollack, J.B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science*, 9:52-74.

[43] Waltz, D.L. (1982). Event shape diagrams. In *Proceedings of Second National Conference on Artificial Intelligence*, pp. 84-87.

10

Examining a Hybrid Connectionist/Symbolic System for the Analysis of Ballistic Signals

CHARLES LIN AND JAMES HENDLER

Department of Computer Science
University of Maryland
College Park, Maryland 20742

1 INTRODUCTION

The field of artificial intelligence (AI) has produced a variety of different problem solving paradigms. Two of the more prominent ones are symbolic AI and connectionism. Some researchers [4] [14] have argued that symbolism and connectionism represent differing computational paradigms, while others have discussed merging the differing strengths and weaknesses of these approaches, as evidenced by the papers in this volume. The general consensus to date seems to be been that symbolic systems are currently better at reasoning tasks and encapsulating expert knowledge, while connectionist systems have been used more successfully in pattern recognition and other perceptual tasks. Interestingly enough, the strengths of symbolic systems correspond to the weaknesses of connectionist systems and vice versa.

To create more powerful AI systems, we must develop systems that can handle both perceptual tasks, such as image or speech recognition, and cognitive tasks, such as reasoning or planning. Since perceptual tasks are the forte of connectionist systems and reasoning tasks are the forte of symbolic systems, a simple and logical step is to combine the two systems into one *hybrid* system. Ideally, a hybrid system overcomes the weaknesses of each of connectionism and symbolic AI, while maintaining their strengths. While tighter couplings between these approaches may be needed to provide a neurocognitively valid model, current hybrids can be used to solve complex engineering problems and to study what capabilities of the different approaches are most useful in which situations.

The ability for a single system to handle perception and expert reasoning makes it useful for engineering applications which demand both facilities. For example, natural language systems which use speech processing require the conversion of sound to words, which is typically done by signal processing or neural network techniques, need to be combined with a syntactic and semantic processor, which are traditonally symbolic. Automated manufacturing requires visual information to be combined with expert reasoning. In image processing, identification of objects using a signal classifier can be improved by knowledge of maps, a model of the object, or environmental information. Thus, these hybrids systems hold promise for use in a wide range of problems.

In this chapter, we discuss the application of a hybrid shell called *SCRuFFY*[1]. The *SCRuFFY* architecture consists of a connectionist component and a symbolic component (see Figure 1). Specifically, a neural network is used to classify input signals. This classification is typically accomplished by breaking up the signal into time segments, and classifying each segment in order. The sequence of classifications is then processed by a temporal pattern matcher which looks for patterns in time and creates facts which the symbolic component, an expert system, can then reason about.

An initial application for *SCRuFFY* was presented in [10]. In it, *SCRuFFY* was used to produce diagnostic messages suggesting a course of action to maintain the normal operating conditions of a simulated underwater robot welder. Based on sensor signal data (taken from a real robot, and programmed into the simulator), *SCRuFFY* determined the operating condition of the welder and, if it determined that there were problems, suggested setpoint changes so as to make corrections.

Based on the initial success of this effort, the *SCRuFFY* architecture was further used in other control tasks. In [2], *SCRuFFY* was used in a chemical process control problem to improve the performance of PIDs. The welding control problem and the chemical process control problem shows that the *SCRuFFY* shell can be used to design hybrid systems which handle control problems. In this chapter, the *SCRuFFY* architecture will be used in a pattern recognition problem. Doing a different type of problem not only demonstrates the broad applicability of the *SCRuFFY* shell to problems which require perception and expert reasoning, but also lets us examine features of these various problem types viz the application of these sorts of techniques.

[1]*S*ignals, *C*onnections, and *Ru*les: *F*un *F*or ever*Y*one

As the goal of this research is to examine the capabilities of the *SCRuFFY* hybrid architecture, the next two sections discuss some of related work in hybrid systems and describe the *SCRuFFY* architecture. Section 3 discusses why the ballistic signal problem was selected, and examines how the hybrid system was used to solve this problem. Sections 5 and 6 describe future work and make concluding remarks.

2 Related Work in Hybrid Systems

While this paper presents one view of a hybrid connectionist-symbolic system, specifically one that uses a neural network and an expert system as its two components, other researchers have also combined neural networks and expert systems as part of a single system. In addition to the approaches described in papers in this volume, several other approaches have been outlined.

In combining neural networks with expert systems, some researchers have built systems that represent expert knowledge, like expert systems do, and yet can learn from examples, like neural networks do. In [12] , Romaniuk and Hall describe a connectionist model which can both encode rules or learn rules via examples. In [5] and [15] , the authors use a connectionist architecture to encode a knowledge base, essentially making a more flexible learning system than explanation-based learning (EBL). Goodman et al [6] [7] [8] extract probablistic rules using information theoretic techniques and encode them in a connectionist-style architecture.

In all of these works, connectionist architectures were used to encode rules. Modifications were, therefore, required to the basic neural network architecture so that these rules could be encoded. Rather than having two distinct components, one symbolic and one connectionist, these systems only have one and thus are not particularly "hybrid."

There is also research whose connectionist and symbolic components are distinct. In [1], Bruja concludes that neural nets are useful tools in pattern recognition, including the recognition of waveforms. Though he does not implement a knowledge-based component into his system, he discusses how such a component could be used. In [9], neural networks and rule-based expert systems are used to control a two-link manipulator in learning to swing (as in swinging a baseball bat). This method is modeled after human learning where certain actions, such as swinging a bat, may be learned initially by rules, but eventually become natural, at which point the rules can then be dispensed. In [3], the authors describe a "federative" system which is composed of connectionist and symbolic components. Given a problem, a manager decomposes the problem into subtasks which are then assigned to the component best suited for solving the subtask. If the component can not solve the task, several components may cooperate to solve the problem.

In [11], Kanal and Raghavan contend that pattern recognition systems require a hybrid approach including one-shot classifiers, neural networks, expert systems and other methods that have been traditionally used alone in pattern recognition, and states that no single approach is best suited for this task. In [16], Ulug presents a hybrid system with a neural network component which classifies measurements of hydralic pressure over time, and then uses an expert system to make diagnostic statements based on the classification.

SCRuFFY shares some characteristics with these hybrid systems but differs in other ways. Unlike [9], where the neural network and the expert system are attempting the same task—namely, swinging a bat—our hybrid system has distinct roles for the neural network component and the expert system component. Like [11], we agree that pattern recognition problems require a hybrid approach. Specifically, expert systems can enhance the pattern recognition ability of neural networks by using domain knowledge. Our system is similar to [16] but uses a temporal pattern matcher to combine the neural network to the expert system, as described in the next section. The neural network classifies consecutive segments of an input signal. The temporal pattern matcher takes this sequence of classifications and looks for temporal patterns. This approach adds more power than examining the entire signal at once (which may not be possible if the signal is being examined online) which is Ulug's approach. Essentially, Ulug's approach is the approach taken by *SCRuFFY* but lacking a temporal pattern matcher which makes it less powerful and less flexible than *SCRuFFY*.

3 DESCRIPTION OF THE *SCRuFFY* ARCHITECTURE

The system described in this paper is a direct application of the *SCRuFFY* shell described in [10]. A block diagram of the hybrid architecture used for *SCRuFFY* can be seen in Figure 1. To begin, the signal obtained from sensor data is preprocessed by a signal processing stage. The digital signal processor makes the task of classification more tractable for the neural network. For example, it can do this by eliminating the noise from the incoming signal or by transforming it into a form where it can be more easily classified. A backpropagation-trained feed-forward neural network then takes the processed signal (or, more typically, a segment of the signal) and produces an output classification.

In order to explain the next stage, the temporal pattern matcher, the neural network needs to be briefly discussed. In particular, the neural network used in *SCRuFFY* uses a method similar to that of NETtalk [13], by moving a time window over the signal. Given an i input neural network, the time window covers i consecutive samples of the input signal from time t to time $t+i-1$. This segment of the signal is classified by the neural network. Then, the window is moved over to sampled from time $t+1$ to $t+i$, and that is classified by the neural network. This process is repeated. The sequence of classifications is presented to the temporal pattern matcher for analysis.

The NETtalk approach allows the *SCRuFFY* system to be used as a control system. In control problems, it is often not possible the entire signal trace (as in [16]) at one time because the input is being received online. Typically, the signal comes from sensors collecting information on the current state of the system. Using the NETtalk approach, a neural network can process the signal as it is being received and decide the current state of the system by examining only a part of the input signal. The expert system then makes control decisions and monitors the effect of the decision from the changes in the input signal.

While neural networks are not the only methods of classification (see [11] for other methods), one advantage is that (i) they can be trained, and (ii) once trained they are quite efficient. Specialized hardware can be made to implement neural networks which can classify much faster than traditional classification systems using statistical methods. This can be an important consideration if the *SCRuFFY* shell is to be used in real applications, such as control, where slow classification may defeat the purpose of using a control system in the first place.

Because the neural network used is numeric in nature—that is, it takes in numeric data as input and produces numeric values as outputs, there is a need to convert this numeric representation into a symbolic representation which a rule-based expert system, or other symbolic system, can reason on. The temporal pattern matcher provides a means for making this conversion. As the name implies, the temporal pattern matcher does not make this conversion based on a single output from the neural network. Rather, it looks for patterns in the consecutive outputs of the neural network. By looking at neural network outputs over time, instead of an output at a single instant in time, the pattern matcher can potentially provide more information to the expert system.

For example, in [10], some of the outputs of the neural network represented normal operation of an underwater robot welder while others represented various types of abnormal operation. The output with the largest value indicated the current state of the underwater robot welder. If a normal output had the largest value, then the robot welder was assumed to be behaving normally. Similarly, if an abnormal output had the largest value, the robot welder was assumed to be behaving abnormally. By looking at all neural networks outputs over time, the temporal pattern matcher could look for some abnormal output increasing over time, but which had not yet become the largest output value. It could then inform the expert system of this temporal pattern. The expert system could then take appropriate preventitive actions so the potential abnormal condition is dealt with before it becomes a problem. If the temporal pattern matcher only provided the current state of information to the expert system, the expert system would not have anticipated the problem, and would only have reacted when the problem occurred.

While the expert system could have looked for patterns instead of the temporal pattern matcher, the responsibility is given to the temporal pattern matcher for *finding* the patterns while the expert system *acts* on the patterns. Furthermore, the expert system is relieved of the responsibility of directly dealing with the numeric output of the neural network. The temporal pattern matcher, therefore, provides the link between the numeric form which neural networks typically produces as output, and the symbolic form which expert systems typically handle.

Finally, the rule-based expert system takes the patterns produced by the temporal pattern matcher, and suggests appropriate actions. In general, the recommendations made by the rule-based component are domain specific. If the hybrid system is being used as a controller, it will make control decisions. It can then judge the effect of these decisions by looking at later patterns. Used in pattern recognition, the expert system uses domain knowledge to improve the recognition or classification done by the neural network.

Comparing *SCRuFFY* to the hybrid systems presented in the previous section, we see that unlike some of the hybrid systems, the neural network and expert system are distinct. Thus, standard neural network and expert system techniques can be immediately applied. Unlike the system presented in [3], the neural network and expert system always work in conjunction, and unlike the system presented in [9], the connectionist component and the symbolic component carry out different kinds of tasks based on the analysis of their relative strengths and weaknesses.

4 ANALYSIS OF BALLISTIC SIGNALS

4.1 INTRODUCTION

The *SCRuFFY* shell has been previously used in control problems, [2] [10]. To show the applicability of this shell to another kind of problems, a problem in pattern recognition was considered. Specifically, a problem from a military domain (provided by the U.S. Army) was considered.

During the testing of experimental artillery, firing artillery can be dangerous if certain abnormal conditions arise in a previous firing. To ensure safety, a sensor records the pressure of the chamber as the artillery is fired. Based on the sensor trace of the firing, an expert can determine whether the firing was normal, and if so, the expert can recommend that it is safe to refire. If the firing is abnormal, the expert will forbid the refiring of the gun and also classify the kind of abnormality. In practice, it may take hours or days for the expert to get a chance to examine the data and make this decision. Thus, long idle times plague this testing process.

To demonstrate the applicability of the *SCRuFFY* shell to this kind of problem, we use the system to makes decisions about the classification of the firings based on sensor information, which normally requires the presence of the human expert. However, the goal of this particular work was more to examine the ability of the *SCRuFFY* shell to handle pattern recognition problems and less to really try a hardcore application development effort on the ballistic analysis problem. Thus, after consultation with experts, we chose to focus our attention on a specific, difficult to analyse, abnormality called *ringing*, rather than trying to recognize a wide range of different abnormalities.. In Figure 10, between sample number 2180 and 2880, there is a small, rapid oscillation. Figure 11 shows only the oscillation. This oscillation is called ringing and is an abnormal condition. Figure 2 shows a normal curve where no ringing has occurred. The *SCRuFFY* shell has the responsibility of identifying regions of ringing and deciding whether this ringing is of sufficient duration to require a warning not to refire.

The input to the hybrid system was an ordered set of data collected from the sensors. The resulting output was a recommendation of whether to refire, and the confidence factor of this recommendation. The hybrid system would also print out the times where it had determined that ringing had occurred, thus facilitating later human checking (in essence, *SCRuFFY* was more like an intelligent filter than an expert diagnostician - due to the explosive results if an improper refiring occured, the goal of the system was to recommend appeal to the human if any ringing at all may have occured).

4.2 THE SIGNAL PROCESSING PHASE

Under ideal circumstances, an expert can look at an input signal segment and classify it. With this same classification information, the neural network could reasonably be assumed to be trainable as to make the same classification and to make reasonable classifications for signal segments that are not in the training set. However, sometimes the signal is either not in a form which makes it easy to classify by the expert, or, there are difficulties in training the neural network to make the correct classifications. Signal processing can be used to aid in the classification done by the neural network. It can be used to remove noise, or convert the signal into a form that is either easier to classify by the neural network or by the expert when creating the training set.

By observing Figure 2 and Figure 10, which are graphs of normal and ringing firings, respectively, it can be seen that the normal curve is essentially a slow-varying curve, while the ringing curve is also a slow-varying curve, but it also has a ringing portion (a small rapid oscillation) during times 2180 to 2880. This ringing does not appear in the normal curve. In signal processing, it is well known that signals which vary slowly are represented by lower frequencies, and signals that oscillate quickly are represented by higher frequencies. By converting the graphs from the time domain to the frequency domain, the lower frequencies can be removed. The graph can then be converted back to the time domain. By selecting a good cutoff frequency (done by visual inspection), the slow varying curve can be eliminated and the oscillations kept. Thus, we used a fourier transform of the original signal, rather than the signal itself, in training and testing the network.

4.3 THE NEURAL NETWORK PHASE

The neural network used in this problem had 20 input nodes, 5 hidden layer nodes, and 2 output layer nodes. There was also a bias node used which had an input that was always set to 1. The output nodes produced an output within the range of -1 to 1.

One of the output nodes represents a normal output while the other represents the ringing output. Although there are several ways of interpreting when an output is normal or ringing, it was decided empirically that when the ringing output had a value that exceeded the normal output by at least 0.4, the neural network should classify the input pattern as ringing. If the normal output exceeded the ringing output by 0.4, the pattern was classified as abnormal. Otherwise, it was classified as undecided. The value of 0.4 was selected so that an appropriate level of noise could be tolerated.

Simply encoding the signal for the neural network was not viable because the neural network would have been prohibitively large if all of the 4500 points of data—the number of data points sampled per test firing —had been presented as inputs. We therefore selected a smaller number of inputs, 20. This number gave enough points to represent about one "period" of oscillation during the ringing portion. A mentioned previously, we used a "window" method similar to the one used in NETtalk. A vector of length 20 consisting of the data collected from time t to time $t + 19$, inclusive, was used as input vectors to the neural network. Thus, the first vector included all samples from time 1 to time 20. The second vector included all samples of data from time 2 to time 21. The last vector consisted of samples from time 4481 to 4500, etc. The vectors were presented in order of increasing time, and the corresponding neural network outputs were produced in the same order.

To train the neural network, the normal curve from Figure 3 and the ringing curve from Figure 11, were broken down into vectors of length 20. Given that only a small percentage of the vectors were preclassified as ringing, equal numbers of normal and ringing vectors were made by repeating the ringing vectors until there were as many ringing vectors as normal ones. A ringing input vector was trained to produce a 1 on the ringing output, and a -1 on the normal output. A normal input vector was trained to produce a 1 on the normal output and a -1 on the ringing output.

Once the training was complete, the number of correct and incorrect classifications were calculated (see Tables 1-4). Tables 1 and 2 were from normal firings — all input vectors were preclassified as normal. Tables 3 and 4 were calculated from firings that showed ringing. The graphs that were considered ringing (Figures 10 and 14) did not have ringing occuring throughout the entire graph. Only the input vectors which included samples that fell in the ringing region were preclassified as ringing. The rest of the input vectors of these graphs were considered normal. For Figure 10, ringing was visually inspected as occurring between times 2180 and 2880. For Figure 14, ringing was visually inspected as occurring between times 2580 and 2880. Any input vector which included samples taken during the times that were ringing were preclassified as ringing, Input vectors taken from the curve in figures 3 and 11 were used to train the neural network. The other two figures (7 and 15) were used to test how well the neural network classified a test set of data (signal data that the network was not trained on).

	Actual classification		
Preclassification	normal	undecided	ringing
normal	97.2%	0.2%	2.6%

Table 1 Of the vectors which were preclassified as normal from Figure 3 (which were all of them), the neural network produced the above classification. The above indicates that of the input vectors which were preclassified as normal, 97.2% were classified as normal by the neural network.

	Actual classification		
Preclassification	normal	undecided	ringing
normal	95.2%	0.31%	4.5%

Table 2 Of the vectors which were preclassified as normal from Figure 7 (which were all of them), the neural network produced the above classification.

Figures 4, 8, 12, and 16 show the outputs of the neural network plotted versus time. The outputs are one of three values: zero, if the input vector was classified as normal; one-half, if the input vector was classified as undecided; one if it was classified as ringing. While Tables 1-4 show that the classification was fairly good, it was not perfect. Ideally, we wanted n consecutive outputs classified as ringing before the entire graph could be considered ringing, with an appropriate choice of n. While we were able to get a block of consecutive

	Actual classification		
Preclassification	normal	undecided	ringing
normal	87.3%	0.45%	12.3%
ringing	1.71%	0.71%	97.6%

Table 3 These were the results of the neural network classification of the input vectors from Figure 11. Some of the input vectors were preclassfied as normal, while others were preclassified as ringing. For example, of the input vectors preclassified as normal, 12.3% were classified as ringing by the neural network.

	Actual classification		
Preclassification	normal	undecided	ringing
normal	82.2%	0.73%	17.1%
ringing	0.80%	0.00%	99.2%

Table 4 The above are the classifications made by the neural network on the input vectors from Figure 15.

neural network output classifications to be *mostly* ringing during the times that were considered ringing, they were not *all* classified as ringing. Hence, we had several short blocks of ringing. To make one large block of ringing, the temporal pattern matcher was used to "smooth" out the results of the neural network and to identify longer blocks of ringing.

4.4 The Temporal Pattern Matcher

In [10], the temporal pattern matcher was used to find patterns such as increasing values of an output (of the neural network) over time, or decreasing values of an output over time. However, such patterns were not particularly useful for this problem. The purpose of the pattern matcher, in our problem, was to improve the expert system's ability to identify ranges of ringing by cleaning up the data produced by the neural network.

Specifically, the pattern matcher looked at a sequence of 40 neural network outputs (from time t to t+39, inclusive). If, in this sequence, 70% of the outputs from the neural network were classified as ringing, then the pattern matcher produced an output of ringing for time t. The effect of this smoothing can be seen in Figures 5, 9, 13, and 17. The temporal pattern matcher produced outputs that were only ringing or normal, using the previous criteria for ringing. The values of 40 outputs and 70% were chosen empirically, inspecting the quality of the results when these numbers were varied. (We did not attempt to define an optimal choice for these parameters, but chose values that produced reasonably good results.)

As the "smoothing" procedure generates long blocks of time classified as ringing, it is much easier to identify when ringing has occurred. Comparing figure 12 to 13, there is a reduction in the amount of blocks of ringing from a numerous amount to four. Of the four, only one block is long. Though it can't be seen in Figure 12, the region between 2100 and 2800 is not a single contiguous block. The smoothing operation creates one contiguous block in Figure 4d, and eliminates many of the smaller ones.

The "smoothing" operation had an additional benefit besides creating long blocks of ringing. It generally improved the accuracy of the classification done by the neural network. That is, more input vectors that were preclassified as normal were classified as normal using this smoothing technique in conjunction with the neural network versus using the neural network alone.[2] It is more important to be able to identify a long segment of ringing because this becomes the basis for deciding whether the abnormality is serious or not, as will be discussed later. A long segment of ringing indicates that a significant amount of ringing has occurred while a short segment might indicate that the ringing is not serious enough to produce a warning. Even if smoothing were to decrease the accuracy of identification by a small amount, as long as the smoothing operation produced a long block of ringing in approximately the correct location where the expert had identified ringing, then the smoothing procedure would have accomplished its task. The results of the classification after smoothing can be seen in Tables 5-8.

However, while the first step of the pattern matcher for this problem was to smooth the results of the neural network,, the main purpose of the pattern matcher is to convert the numeric output into facts which the expert system can manipulate. In this case, it takes the graphs from Figures 5, 9, 13, and 17 and produces facts of the form, `(ring` *start time end time*`)`. The fact says that every single output between the start time and the end time was classified

[2]The accuracy also improved for the vectors which were preclassified as ringing. However, as the neural network had already done a fairly accurate job of classification, the improvement in accuracy is not significant.

	Actual classification	
Preclassification	normal	ringing
normal	100.0%	0.0%

Table 5 This is the classification made by the "smoothing" procedure on Figure 4. It was decided that only two classifications were needed: normal and ringing.

	Actual classification	
Preclassification	normal	ringing
normal	99.5%	0.5%

Table 6 This is the classification made by the "smoothing" procedure on Figure 8.

	Actual classification	
Preclassification	normal	ringing
normal	94.2%	5.8%
ringing	0.0%	100.0%

Table 7 This is the classification made by the "smoothing" procedure on Figure 12.

as ringing by the smoothing procedure. Facts are generated for all such

	Actual classification	
Preclassification	normal	ringing
normal	97.6%	2.4%
ringing	9.4%	90.6%

Table 8 This is the classification made by the "smoothing" procedure on Figure 16.

consecutive sequences of ringing for a given graph, one fact for each block found. If no ringing occurred in the entire graph, then the pattern matcher produces the fact `(none)`, indicating no ringing occurred. These facts are passed to the expert system as they are discovered.

4.5 THE EXPERT SYSTEM PHASE

Since the problem of recognizing ringing in signals generated by ballistic firings is primarily one in pattern recognition and because the authors had no additional domain knowledge from which to work with, the expert system is fairly simple. Its main task is to identify whether the graph of a firing has ringing or not, to determine whether to refire the gun which produced this graph, and to indicate the times it believes ringing has occurred. The expert system used in our problem was written in CLIPS, which is a forward-chaining rule-based expert system similar to OPS5, but written in C. (In fact, this entire system is written in C, and runs on a Sun Sparc station.)

As previously mentioned, the temporal pattern matcher produces facts of the form `(ring` *start time end time*`)`. We call this span of time a *ringing block.* The length of the ringing block is the difference between the end time and the start time.

The expert system takes these facts and divides them into three categories. Either the ringing block is longer than 100, between 10 and 100, or between 1 and 10. If there are blocks of length 100 or more, the expert system concludes that there is definite evidence for ringing and advises not to refire the gun. It also prints out the times where it believes ringing occurred. If a block of this length is found, then any block of ringing with length less than 100 is ignored since it has already been concluded that ringing has occurred.

If there are no blocks of ringing longer than 100, but there are ringing blocks of length 10 to 100 or 1 to 10, then the times of the ringing blocks are again printed. The longer the ringing, the more likely the advice is not to fire. Only when there are no ringing blocks at all will the expert system advise that it is safe to refire the gun. A resulting run of the CLIPS expert system for all four curves is seen in Figures 18, 19, 20, and 21.

However, this result neither showed the real power of the expert system, nor corresponded to the real analysis in one important way. It turns out that certain oscillations in the curve do *not* correspond to ringing, and should not prevent the firing of the gun. If the oscillation occured early in the firing (represented by the original upcurve of the graphs shown previously), then the oscillation was not really ringing. However, the fourier method and the neural network could not easily be designed so as to recognize the difference, thus the expert system was needed to eliminate the spurious reports of ringing.

A rule was added to the expert system which determined whether the evidence of ringing occurred during the upslope of the curve or the downslope. If ringing occurred on the upslope, it was considered to be less problematic, and was ignored. By adding this rule, the expert system concluded (see Figure 22) that it was safe to refire, whereas in Figure 19 which does not use this rule, the expert system advises not to refire. Figures 20 and 22 are run on the same input with the only difference being the change in rules of the expert system. This shows that expert knowledge can be used to identify unusual cases which would normally be incorrectly classified by the neural network.

5 FUTURE WORK

The work described in this paper was, essentially, a proof of concept for the use of the *SCRuFFY* hybrid system on the ballistic analysis task. The logical next step is to increase the difficulty of this problem. As described, only one kind of abnormal condition—ringing—was considered. There were, however, several other kinds of abnormal conditions. With these additional abnormal conditions, issues such as considering whether one neural network should make all the classifications, or whether there should be a separate neural network for each of the different abnormalities have to be addressed. Also, the expert system now has a more complicated task because it must decide which of several abnormalities is occurring. How the hybrid *SCRuFFY* shell will scale as the difficulty of the problem increases is still the subject of future research.

Additionally, we are pursuing research to determine what sorts of problems are best suited to the hybrid approach discussed in this paper, and what extensions are needed. The original temporal pattern matcher [10] was based on James Allen's logic of temporal intervals. For the pattern recognition task, it was clear we needed another approach, such as the smoothing procedure described previously. We believe the temporal pattern matching approach is critical to the success of hybrid systems, but believe more research is needed to determine exactly what the language for the temporal patterns should be.

6 CONCLUSION

To use hybrid models effectively, it is necessary to examine the kinds of problems that connectionist and symbolic systems are suited to solving. By identifying these tasks, we can find problems that are easily handled by hybrid systems, while challenging the capabilities (or scaling) of either neural or symbolic approaches. As an example, we have attempted to demonstrate that hybrid systems can be effectively used for a combination pattern-recognition and decision-making problem. Specifically, we used the hybrid system to classify a signal taken from the firing of a gun as either normal or ringing.

As we have shown, hybrid systems represent one way of increasing the power of connectionist and symbolic systems, taking advantage of each of their strengths. When used in solving pattern classification problems, the expert system is useful if domain knowledge is readily available and can augment the classification work done by the neural network. For this task, a non-hybrid approach would either have produced an inferior result or required significantly more training (if a neural network) or significantly more rules (if a purely rule-based approach). Expert Systems have not typically been well-suited for performing pattern recognition for sensory inputs, one of the reasons for the popularity of neural networks in pattern recogntion. However, as discussed earlier, a standard connectionist system has significant trouble in taking advantage of domain knowledge, and in making decisions such as when ringing was significant and whether the gun should be refired. The hybrid approach allowed us to use the strengths of both and thereby to improve the solution.

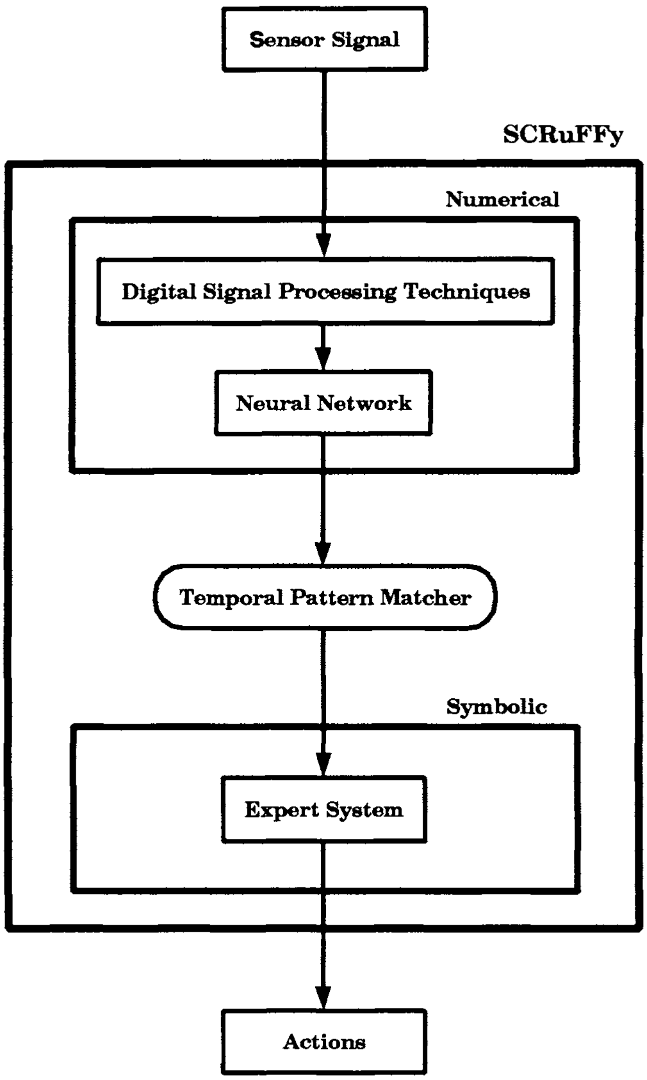


Figure 1 A block diagram of the *SCRuFFY* system. Sensor signals serve as the input. DSP techniques and neural networks form the numerical manipulation of the data. The temporal pattern matcher converts the numeric output of the neural network to a symbolic output which the expert system then decides on appropriate output actions.

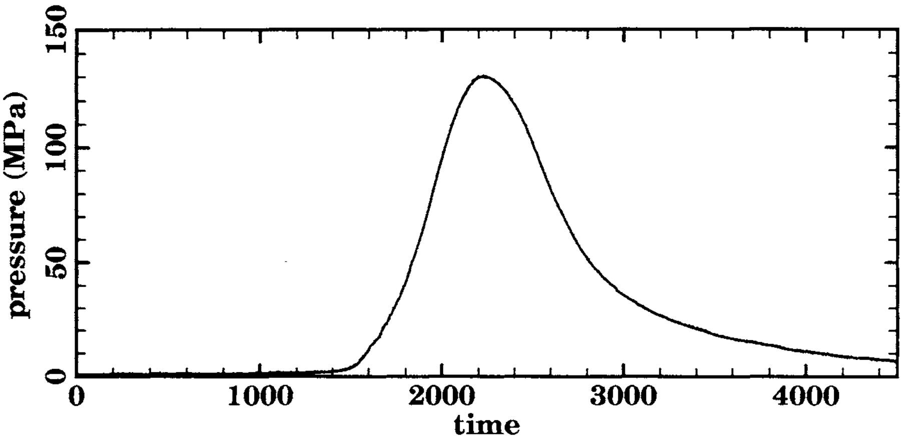


Figure 2 Graph of a normal firing. Neural network was trained on this curve. x-axis refers to sample number (time). y-axis refers to pressure measured in MPascals. No ringing occurs in this curve.

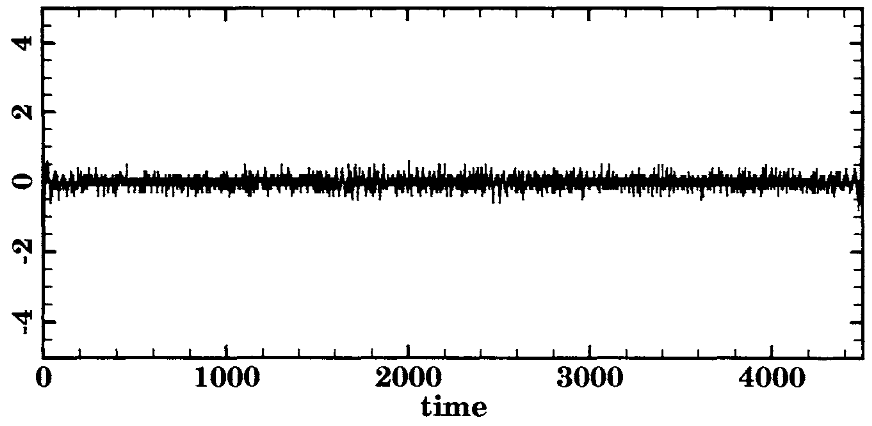


Figure 3 This is curve from Figure 2 with the lower frequencies removed. The y-axis refers to sample number (time). Notice there are no large oscillations (as in Figure 11) which would be an indication of ringing.

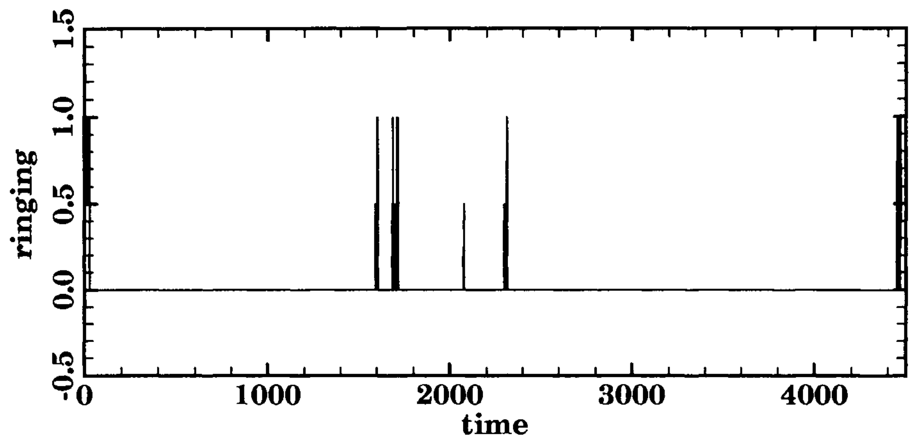


Figure 4 The output of the neural network on Figure 3. The x-axis refers to sample number (time). The y-axis has three discrete values. 0 means no ringing. 0.5 means uncertain. 1 means ringing.

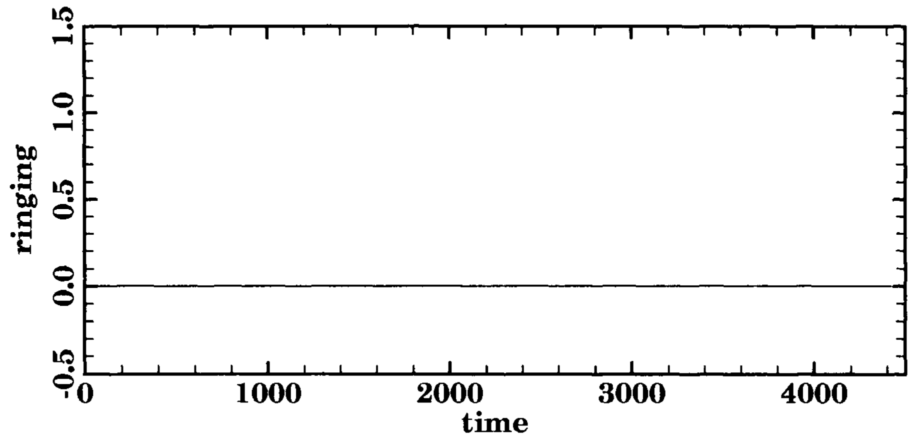


Figure 5 This is Figure 4 after "smoothing". The spikes that might have indicated ringing have been removed. The axes are the same as in Figure 4.

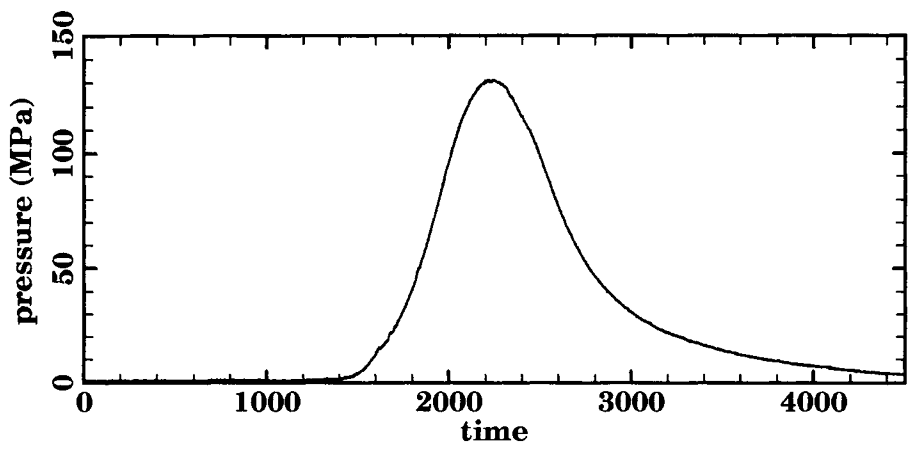


Figure 6 Graph of a normal firing. Neural network was *not* trained on this curve. x-axis refers to sample number (time). y-axis refers to pressure measured in MPascals. No ringing occurs in this curve.

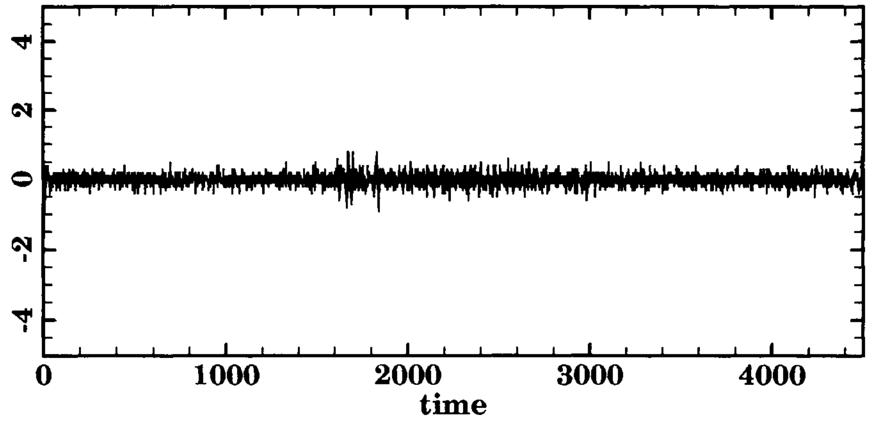


Figure 7 This is curve from Figure 6 with the lower frequencies removed. The y-axis refers to sample number (time). Notice there are no major oscillations (as in Figure 11) which would be an indication of ringing.

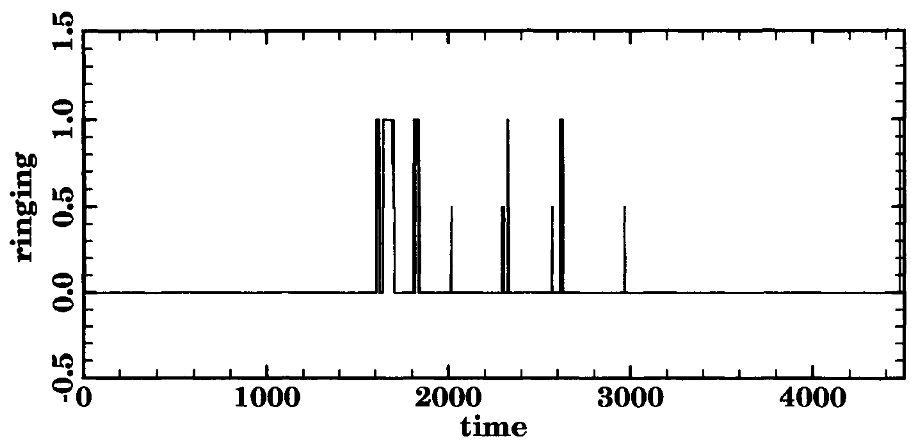


Figure 8 The output of the neural network on Figure 7. The x-axis refers to sample number (time). The y-axis has three discrete values. 0 means no ringing. 0.5 means uncertain. 1 means ringing.

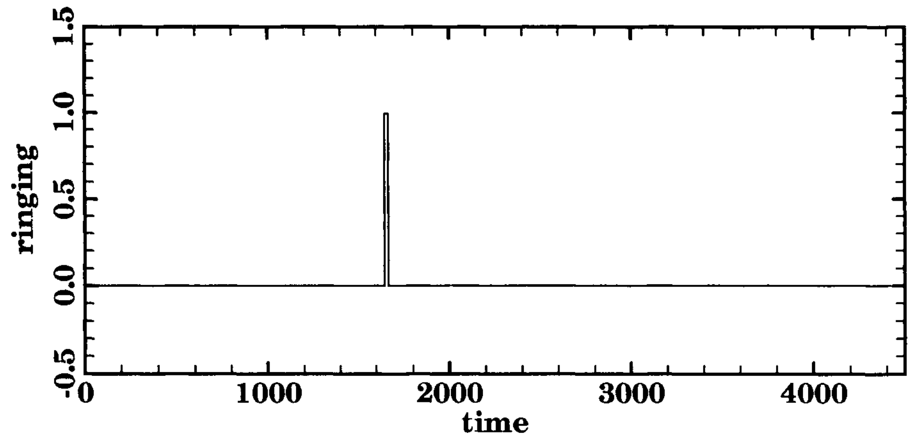


Figure 9 This is Figure 8 after "smoothing". Most of the spikes from Figure 8 that might have indicated ringing have been removed. The axes are the same as in Figure 8.

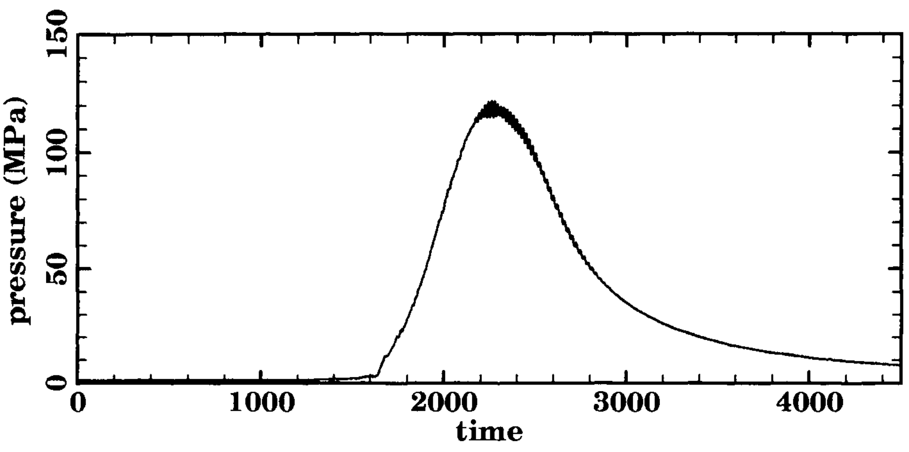


Figure 10 Graph of an abnormal firing. Ringing has occurred in this curve. Neural network was trained on this curve. x-axis refers to sample number (time). y-axis refers to pressure measured in MPascals.

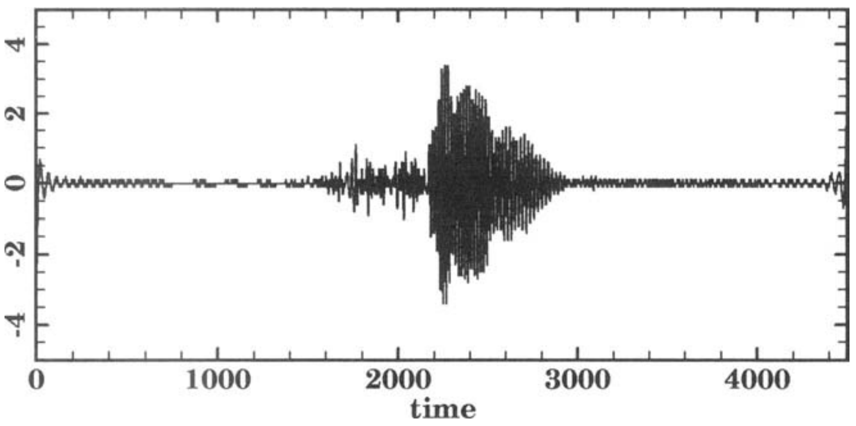


Figure 11 This is curve from Figure 10 with the lower frequencies removed. The y-axis refers to sample number (time). Notice there are large spikes between times 2180 and 2880 which indicates ringing.

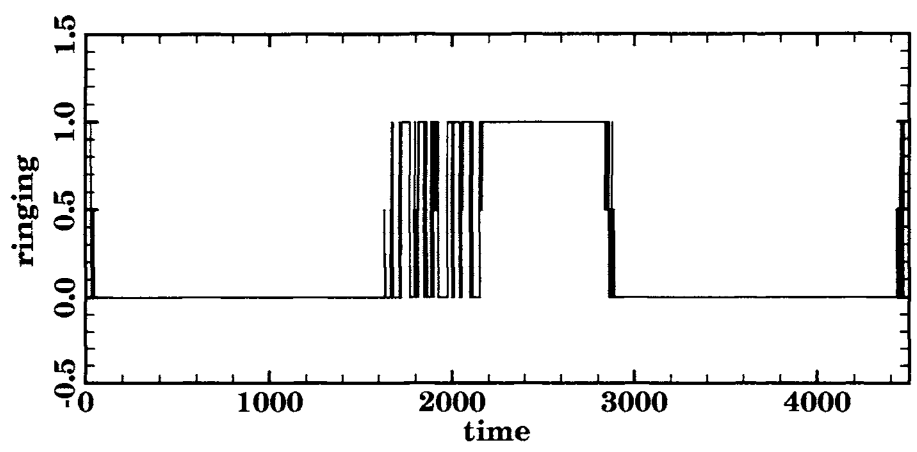


Figure 12 The output of the neural network on Figure 11. The x-axis refers to sample number (time). The y-axis has three discrete values. 0 means no ringing. 0.5 means uncertain. 1 means ringing. Notice that between times 2000 to 3000 there is a solid range with a value of 1 which is also the times when ringing occurs.

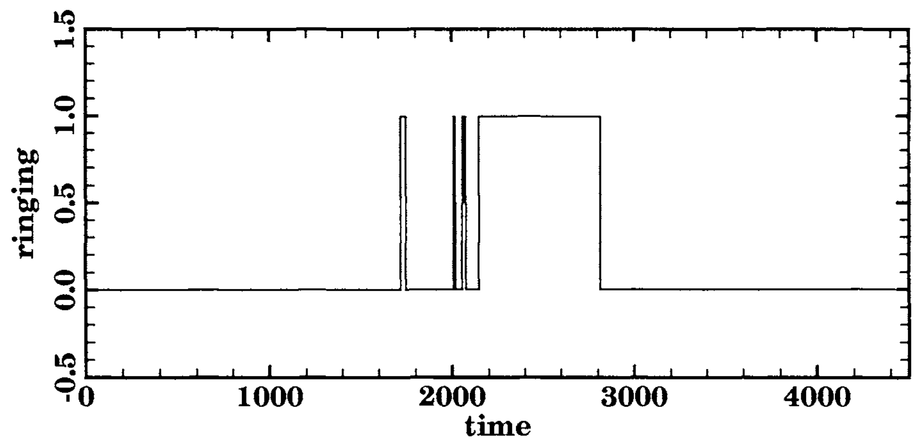


Figure 13 This is Figure 12 after "smoothing". Notice that much of the "noise" in Figure 10 has been cleaned up. Between times 2100 and 2800, there is one large block of ringing which the expert system will look for to determine ringing. The axes are the same as in Figure 12.

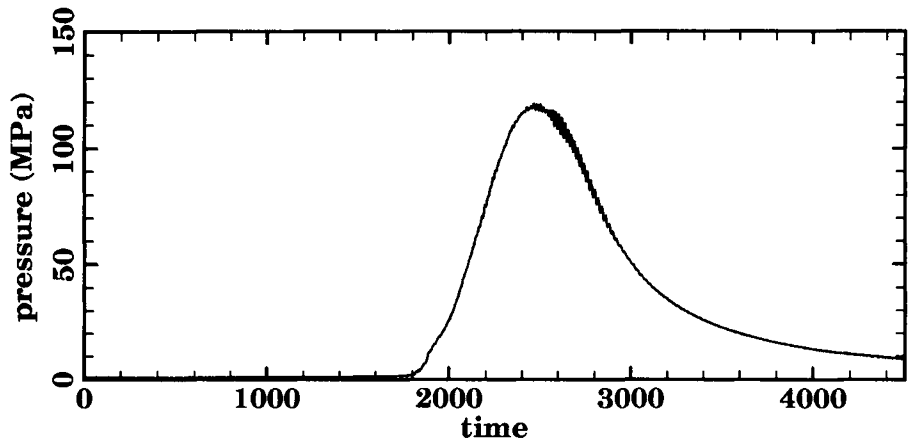


Figure 14 Graph of an abnormal firing. Ringing occurs in this curve between times 2580 and 2880. The neural network was *not* trained on this curve. x-axis refers to sample number (time). y-axis refers to pressure measured in MPascals.

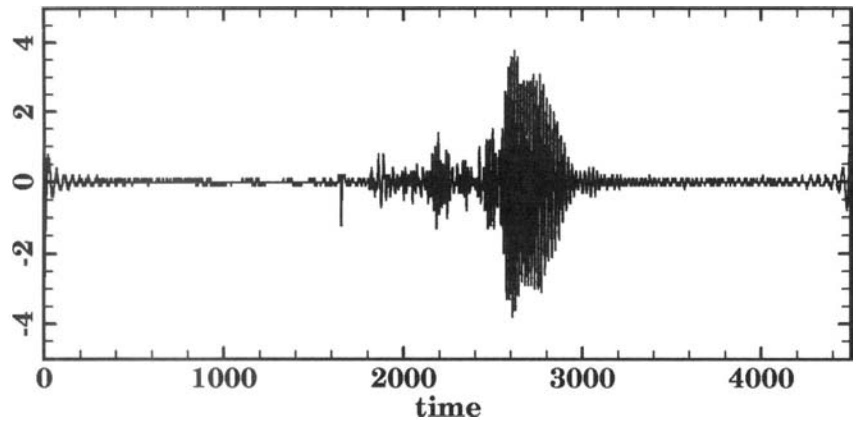


Figure 15 This is curve from Figure 14 with the lower frequencies removed. The y-axis refers to sample number (time). Notice there are large oscillations between times 2580 and 2880 which indicate ringing.

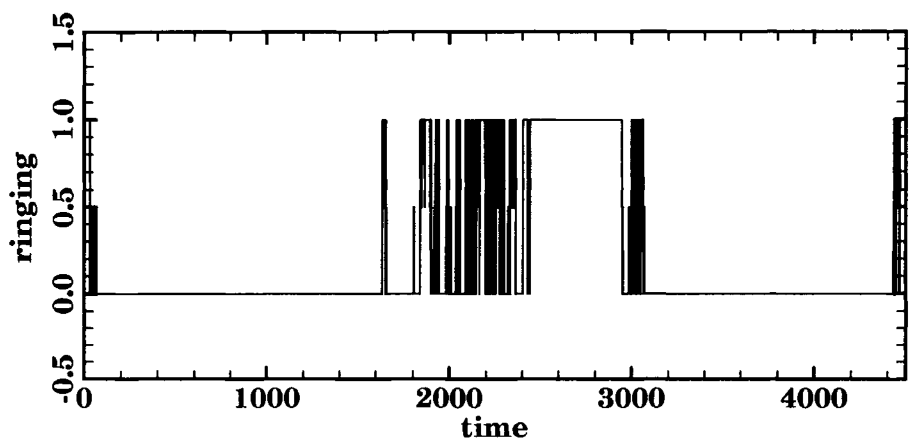


Figure 16 The output of the neural network on Figure 15. The x-axis refers to sample number (time). The y-axis has three discrete values. 0 means no ringing. 0.5 means uncertain. 1 means ringing. Notice that between times 2600 to 3000 there is a solid range with a value of 1 which is also the times when ringing occurs.

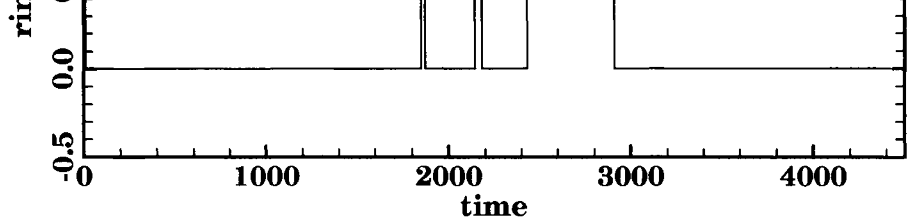


Figure 17 This is Figure 16 after "smoothing". Notice that much of the "noise" has been cleaned up. Between times 2400 and 2800, there is one large block of ringing which the expert system will look for to determine ringing. The axes are the same as in Figure 16.

```
CLIPS> (run)
(run)
Type in file to be read ==> tt_norm1.clips

You may fire.  There is negligible evidence of ringing.

3 rules fired
```

Figure 18 This is a sample CLIPS run based on the output produced by the temporal pattern matcher on Figure 5. No ringing has been found, so the expert system indicates refiring is safe.

```
CLIPS> (run)
(run)
Type in file to be read ==> tt_norm2.clips

Don't fire without more information!
Moderate evidence of ringing between time 1645 and 1665
Length of ringing: 20

4 rules fired
```

Figure 19 This is a sample CLIPS run based on the output produced by the temporal pattern matcher on Figure 9. A little ringing has been found, so the expert system indicates a moderate warning not to refire.

```
CLIPS> (run)
(run)
Type in file to be read ==> tt_ring2.clips

Do not fire!
High evidence of ringing between time 2148 and 2814
Length of ringing: 666

8 rules fired
```

Figure 20 This is a sample CLIPS run based on the output produced by the temporal pattern matcher on Figure 13. Definite indications of ringing has been found, so the expert system advises not to refire.

```
CLIPS> (run)
(run)
Type in file to be read ==> tt_ring1.clips

Do not fire!
High evidence of ringing between time 2434 and 2908
Length of ringing: 474

7 rules fired
CLIPS> (exit)
(exit)
```

Figure 21 This is a sample CLIPS run based on the output produced by the temporal pattern matcher on Figure 17. Definite indications of ringing has been found, so the expert system advises not to refire.

```
CLIPS> (run)
(run)
Type in file to be read ==> tt_norm2.clips

Evidence of ringing between time 1645 and 1665
Being removed from consideration because it
falls during times 0 and 1700 where there should
should be no ringing.

You may fire.  There is negligible evidence of ringing.

5 rules fired
```

Figure 22 This is a sample CLIPS run based on the output produced by the temporal pattern matcher on Figure 9. An additional rule has been added to discount ringing when it occurs during the upslope of a curve. The expert system advises that refiring is safe, but also indicates that it is ignoring ringing on the upslope.

REFERENCES

[1] I. Bruja (1992). Neural Net Approach to Recognition of Waveforms, *Artificial Intelligence and Tutoring Systems for Teaching and Learning*, pp. 43–63, London: Ellis Horwood Limited.

[2] Y. Cui, J. Hendler, H. Su, and T. McAvoy. (in press). Improving PID controllers with a Neural Network. Tech. rep., Institute of Systems Research, University of Maryland, 1993.

[3] J. Dunker, A. Scherer, and G. Schlageter. (1992). Integrating Neural Networks into a Distributed Knowledge Base. In *Int. Conf. on AI, Expert Systems and Neural Language.*

[4] M. Dyer. (1988). Symbolic NeuroEngineering for Natural Language Processing: A Multilevel Research Approach. Tech. Rep. UCLA-AI-88-14, Computer Science Dept., University of California, Los Angeles.

[5] G.M. Scott, J.W. Shavlik and W.H. Ray. (1991). Refining PID Controllers using Neural Networks, In J. Moody, S. Hanson and R. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4*, pp. 555–562, Morgan Kaufmann: San Mateo, CA.

[6] R. M. Goodman, C. M. Higgins, J. W. Miller, and P. Smyth. (1992). Rule-Based Neural Networks for Classification and Probability Estimation. *Neural Computation*, No. 4, pp. 781–804.

[7] R. M. Goodman, J. W. Miller, and P. Smyth (1989). An Information Theoretic Approach to Rule-Based Connectionist Expert System, *Advances in Neural Information Processing Systems 1*, pp. 256–263, San Mateo, CA: Morgan Kaufmann.

[8] H. K. Greenspan, R. Goodman, and R. Chellappa. (1992). Combined Neural Network and Rule-Based Framework for Probabilistic Pattern Recognition and Discovery, *Advances in Neural Information Processing Systems 4*, pp. 444–451, San Mateo, CA: Morgan Kaufmann.

[9] D. A. Handelman, S. H. Lane, and J. J. Gelfand. (1990). Integrating Neural Networks for Intelligent Robot Control. *IEEE Control System Magazine*, pp. 77–87.

[10] J. Hendler and L. Dickens. (1992). Integrating Neural Network and Expert Reasoning: An Example. *Proceedings of AISB*, pp. 109–116.

[11] L. Kanal and S. Raghavan. (1992). Hybrid Systems-A Key to Intelligent Pattern Recognition. *International Joint Conference of Neural Networks*, Vol. IV of IV, June 7-11.

[12] S. G. Romaniuk and L. O. Hall. (1993). SC-net: A Hybrid Connectionist, Symbolic System." *Information Sciences.*

[13] T. Sejnowski and C. R. Rosenberg. (1986). NETtalk: a parallel network that learns to read aloud. Tech. Rep. EECS-86/01, Johns Hopkins Univ.

[14] P. Smolensky. (1990). Connectionist AI, Symbolic AI, and the Brain. *AI Review.*

[15] G. G. Towell, J. W. Shavlik, and M. O. Noordweier. (1990). Refinement of Approximate Domain Theories by Knowledge-Based Neural Networks. In *Proceedings, AAAI-90*, pp. 861-866.

[16] M. E. Ulug. (1989). A Hybrid Expert System Combining AI Techniques with a Neural-Net. In *Proceedings of the Second International Conference on Industrial and Engineering Applications of AI and Expert Systems*, pp. 305-309.

PART IV

COMMENTARIES

Part IV: Commentaries

- Chapter 11 (by Vasant Honavar) attempts to explore questions regarding fundamental similarities and differences between symbolic systems and connectionist systems.
- Chapter 12 (by Michael Dyer) reviews connectionist symbolic processing models with respect to natural language processing.

11

Symbolic Artificial Intelligence and Numeric Artificial Neural Networks: Towards a Resolution of the Dichotomy

VASANT HONAVAR

Department of Computer Science
Iowa State University
Ames, Iowa 50011

1 INTRODUCTION

The attempt to understand intelligence entails building theories and models of brains and minds, both natural as well as artificial. From the earliest writings of India and Greece, this has been a central problem in philosophy. The advent of the digital computer in the 1950's made this a central concern of computer scientists as well (Turing, 1950). The parallel development of the theory of computation (by John von Neumann, Alan Turing, Emil Post, Alonzo Church, Charles Kleene, Markov and others) provided a new set of tools with which to approach this problem — through analysis, design, and evaluation of computers and programs that exhibit aspects of intelligent behavior — such as the ability to recognize and classify patterns; to reason from premises to logical conclusions; and to learn from experience.

In their pursuit of artificial intelligence and mind/brain modelling, some wrote programs that they executed on serial stored-program computers (e.g., Newell, Shaw and Simon, 1963; Feigenbaum, 1963); Others had more parallel, brain-like networks of processors (reminiscent of today's connectionist networks) in mind and wrote more or less precise specifications of what such a realization of their programs might look like (e.g., Rashevsky, 1960; McCulloch and Pitts, 1943; Selfridge and Neisser, 1963; Uhr and Vossler, 1963); and a few took the middle ground (Uhr, 1973; Holland, 1975; Minsky, 1963; Arbib, 1972; Grossberg, 1982; Klir, 1985).

It is often suggested that two major approaches have emerged — symbolic artificial intelligence (SAI) and (numeric) artificial neural networks (NANN or connectionist networks) and some (Norman, 1986; Schneider, 1987) have even suggested that they are fundamentally and perhaps irreconcilably different. Indeed it is this apparent dichotomy between the two apparently disparate approaches to modelling cognition and engineering intelligent systems that is responsible for the current interest in computational architectures for integrating neural and symbolic processes. This topic is the focus of several recent books (Honavar and Uhr, 1994a; Goonatilake and Khebbal, 1994; Levine and Aparicioiv, 1994; Sun and Bookman, 1994). This raises some important questions: What exactly are symbolic processes? What do they have to do with SAI? What exactly are neural processes? What do they have to do with NANN? What (if anything) do SAI and NANN have in common? How (if at all) do they differ? What exactly are computational architectures? Do SAI and NANN paradigms need to be integrated? Assuming that the answer to the last question is yes, what are some possible ways one can go about designing computational architectures for this task? This chapter is an attempt to *explore* some of these fundamental questions in some detail.

This chapter argues that the dichotomy between SAI and NANN is more perceived than real. So our problems lie first in dispelling misinformed and wrong notions, and second (perhaps more difficult) in developing systems that take advantage of both paradigms to build useful theories and models of minds/brains on the one hand, and robust, versatile and adaptive intelligent systems on the other. The first of these problems is best addressed by a critical examination of the popular conceptions of SAI and NANN systems along with their philosophical and theoretical foundations as well as their practical implementations; and the second by a judicious theoretical and experimental exploration of the rich and interesting space of designs for intelligent systems that integrate concepts, constructs, techniques and technologies drawn from not only SAI (Ginsberg, 1993; Winston, 1992) and NANN (McClelland and Rumelhart, 1986; Kung, 1993; Haykin, 1994; Zeidenberg, 1989), but also other related paradigms such as statistical and syntactic pattern recognition (Duda and Hart, 1973; Fukunaga, 1990; Fu, 1982; Miclet, 1986)), control theory (Narendra and Annaswamy, 1989) systems theory (Klir, 1969), genetic algorithms (Holland, 1975; Goldberg, 1989; Michalewicz, 1992) and evolutionary programming (Koza, 1992). Exploration of such designs should cover a broad range of problems in perception, knowledge representation and inference, robotics, language, and learning, and ultimately, integrated systems that display what might be considered human-like general intelligence.

2 Shared Foundations of SAI and NANN

This section makes clear that the fundamental philosophical assumptions and scientific hypotheses that have shaped both SAI and NANN research are identical. The shared foundations of SAI and NANN guarantee that there can be no fundamental incompatibility between the two paradigms for engineering intelligent systems or for modelling minds/brains.

2.1 SAI and NANN Share the Same Working Hypotheses

The fundamental working hypothesis that has guided most of the research in artificial intelligence as well as the information-processing school of psychology is rather simply stated: *Cognition, or thought processes can, at some level, be modelled by computation.* The philosophical roots of this hypothesis can be traced at least as far back as the attempts of Helmholtz, Leibnitz and Boole to explain thought processes in terms of mechanical (or in modern terms, algorithmic or computational) processes. This has led to the *functional* view of intelligence which is shared explicitly or implicitly by almost all of the work in SAI as well as NANN. Newell's *physical symbol system hypothesis* (Newell, 1980), Fodor's *language of thought* (Fodor, 1976), Minsky's *society of mind* (Minsky, 1986), Holland's *classifier systems* (Holland, 1986), and most neural network models (McClelland and Rumelhart, 1986; Kung, 1993; Haykin, 1994; Zeidenberg, 1989) are all specific examples of this functional view. In this view, intelligence can be characterized abstractly as a functional capability independent of any commitments as to the specific physical substrates that support the functions in question.

The primary means of describing the behavior of intelligent systems (be they natural or artificial) within the SAI paradigm is in terms of their having *knowledge* and behaving in light of that knowledge. This is the so-called *knowledge-level* description (Newell, 1990). But it is important to remember that descriptions at the knowledge-level represent just one of the many alternatives available. The choice of what description to use in modelling intelligence, as in science in general, must be based on pragmatic considerations as determined by aspects of the phenomena being modelled and the sorts of explanations being sought. Satisfactory accounts of system behavior often make use of multiple levels of description along with the necessary means of mapping descriptions at one level into descriptions at adjacent levels.

Perhaps not so obvious is the fact that exactly the same functional view of intelligence is at the heart of current approaches to mimic intelligent behavior within the NANN paradigm, as well as the attempts to understand brain function using the techniques of computational neuroscience and neural modelling. The earliest work on neural networks by Rashevsky (1960), McCulloch and Pitts (1943) and Rosenblatt (1962)nociteros62 — from which many of today's NANN models are derived — illustrates this point rather well. So does the emphasis on *computational* models in the recent book on this topic by Patricia Churchland and Terrence Sejnowski (1992) suggestively titled *The Computational Brain* (This is not to suggest that brain modelling can ignore the particular biological substrates that realize the computations in question but just that the computational characterization of what the brains do provides a useful class of explanations and predictions of mental phenomena). It is important to note that NANN models or theories of intelligence are stated in terms of abstract computational mechanisms just as their SAI counterparts. The differences (if any) from SAI are primarily in terms of the (often unstated) preference for functional descriptions of intelligent systems at a different level of detail using a different set of primitives.

Some have (somewhat misleadingly) used the term *neural level* to refer to such descriptions. Today's NANN models are almost certainly grossly oversimplified caricatures of biological brains (Shepherd, 1989; 1990). It is far from clear that NANN are more suited to modelling brains any more than SAI; Descriptions in terms of rules, tokens, and automata (typically associated with SAI systems) offer extremely useful descriptions of biological neural circuits at the cellular and molecular levels (Cooper, 1990). (More on this later).

It should be clear from the discussion above that both SAI and NANN paradigms essentially offer two different *description languages* for describing systems in general and intelligent systems — minds/brains (be they natural or artificial) — in particular. As pointed out by Chandrasekaran and Josephson (1994), the commitment of most SAI researchers to biology in describing intelligence does not typically go beyond the knowledge level. Although perhaps not as obvious is the fact that an analogous situation holds for NANN models. NANN researchers pick out some interesting or relevant aspects of biological phenomena, and then proceed to formulate an abstract functional model (using abstract models of neurons) for the selected aspects of the phenomena chosen. The abstract descriptions in both cases are usually stated in sufficiently general languages. One thing we know for certain is that such languages are all equivalent (see below). This provides absolute assurance that particular physical implementations of systems exhibiting mind/brain-like behavior can be described using the language of SAI or NANN irrespective of the physical medium (biological or silicon or some other) that is used in such an implementation. And the choice of the language should (as it usually is, in science in general) be dictated by pragmatic considerations.

2.2 SAI and NANN Rely on Equivalent Models of Computation

Turing was among the first to formalize the common-sense notion of computation in terms of execution of what he called an *effective procedure* or an algorithm. In the process, he invented a hypothetical computer — the *Turing machine*. The behavior of the Turing machine is governed by an algorithm which is realized in terms of a *program* or a finite sequence of instructions. Turing also showed that there exists a *universal* Turing machine (essentially a general purpose stored program computer with potentially infinite memory) — one that can *compute* anything that any other Turing machine could possibly compute — given the necessary program as well as the data and a means for interpreting its programs. Several alternative models of computation were developed around the same time including *lambda calculus* of Church and Rosser, *Post productions*, *Markov algorithms*, *Petri nets*, and *McCulloch-Pitts neural networks*. However, all of these models (given potentially infinite memory) were proved exactly equivalent to the Turing Machine. That is, any computation that can be described by a finite program can be programmed in any general purpose language or on any Turing-equivalent computer (Cohen, 1986). (However, a program for the same computation may be much more compact when written in one language than in some other; or it may execute much faster on one computer than some other). But the provable equivalence of all general purpose computers and languages assures us that *any* computation — be it numeric or symbolic — can be realized, in principle, by both SAI as well as NANN systems.

Given the reliance of both SAI and NANN on equivalent formal models of computation, the questions of interest have to do with the identification of particular subsets of Turing-computable functions that model various aspects of intelligent behavior given the various design and performance constraints imposed by the physical implementation media at our disposal.

3 KNOWLEDGE REPRESENTATION REVISITED

Knowledge representation and inference are perhaps among the most central research issues in the integration of SAI and NANN paradigms for modelling cognitive phenomena and engineering intelligent systems. This is evident from the fact that almost all the recent books on the integration of SAI and NANN paradigms (Honavar and Uhr, 1994a; Levine and Aparicioiv, 1994; Sun and Bookman, 1994) have devoted several chapters to this topic. It is therefore worth clarifying some basic issues about knowledge representation.

It is generally accepted in artificial intelligence and cognitive science that knowledge has to be *represented* in some form in order for it to be used. This is free of any commitment as to how a particular piece of knowledge is internally represented. However, implicit in this view is a commitment to use *some* language (e.g., first order logic, production rules, lambda calculus or LISP) to express and manipulate knowledge. Expressions in any such language can be syntactically transformed into any other sufficiently expressive language — this follows from the universality of the Turing framework. This is tantamount to saying that systems that use knowledge are simultaneously describable at multiple levels of description. And systems (such as living brains or robots) that exist in the physical world would have physical descriptions — just as the behavior of a computer can be described at an abstract level in terms of data structures and programs, or in terms of machine language operations that are carried out (thereby making the function of the hardware more transparent) or in terms of the laws of physics that describe the behavior of the physical medium which is used to construct the hardware.

Note that this view is entirely consistent with that of Churchland and others (Churchland, 1986; Churchland and Sejnowski, 1992) who have advocated the search for explanations of cognition at multiple levels. It is also important to not lose sight of the fact that such a system is embedded within an external environment with which it interacts in a closed loop fashion through sensors and effectors and its body of knowledge is *about its* environment, *its* goals, *its* actions. This is the essence of *grounding* (see below).

3.1 NATURE OF KNOWLEDGE REPRESENTATION

Given the central role played by knowledge representation in functional approaches to understanding and engineering intelligence, the *nature of representation* is among one the most fundamental questions in artificial intelligence and cognitive science. Some insight into this question can be obtained by considering a concrete example. A common way to represent knowledge (at least in SAI) is with logic (Genesereth and Nilsson, 1987). It is worth emphasizing that logic is not the knowledge itself; it is simply a way of representing knowledge. (However, logic can be viewed as a form of meta-level knowledge about how to represent and reason with knowledge.) What logic enables us to do is represent the knowledge possessed by an agent using a finite set of logical expressions plus a process (namely, the inference rules of logic) for generating a (potentially unlimited) set of other logical expressions that are part of the agent's knowledge. Thus if we represented an agent's knowledge in the form of expressions a and b, and if $a \wedge b \models c$, the agent has (implicit) knowledge of c even though c was not part of the (explicit) representation. In fact, first order logic is universal in that it is powerful enough to represent essentially any knowledge that can be captured by a formal system. However, for certain types of knowledge to be used for certain purposes (e.g., knowledge of the sort that is captured by maps of some geographical region or a city), first order logic representation may be awkward, indirect, or overly verbose.

If on the other hand, we were to choose a different way of representing knowledge of an agent, one which did not permit any logical deduction, then the agent's knowledge could be limited to those expressions that were explicitly included in the representation. Such a representation is in essence, simply a lookup table for the expressions in question. Thus, (for lack of a better term), the *knowledge content* of a representation may be limited by restricting either the inferences allowed, the form of the expressions that may be included (that is, limiting the expressive power), or both. Indeed, it is often necessary to impose such limits on the power of representation in order to make their use computationally feasible (perhaps at the expense of logical soundness, completeness, or both).

In order for any system to serve the role of a representation (as used in most artificial intelligence and cognitive science theories) it must include: an encoding process that maps the physical state of the external environment into an internal state; processes that map transformations of the physical state of the external environment into appropriate (internal) transformations of the internal state; a decoding process that maps an internal state into a physical state of the external environment — all subject to the constraint that the result of decoding the result of application of internal transformations of an internal state (obtained from encoding a physical state of the environment) is the *same* as the result of directly transforming the physical state of the external environment. (This is perhaps a stronger requirement than is necessary — most likely influenced by the emphasis on logic. It is easy to see several ways of relaxing this constraint

— by allowing the correspondence to be only *approximate* instead of exact, or attainable only with a certain probability. It must also be mentioned that not everyone agrees with this view of representation through encoding; See Bickhard, 1993 for a dissenting view). In short, representations are caricatures of selected aspects of an agent's environment that are operationally useful to the agent. Thus, certain mental operations on the representation can be used to predict the consequences of performing corresponding physical actions on the environment in which the agent operates.

Note that the internal transformations may be performed using LISP programs or production systems of SAI or by a suitably structured NANN. (Note however that the encoding and decoding processes are not purely symbolic because they have to deal with transduction or grounding. Also worth noting is the fact that most systems, be they SAI or NANN only simulate transduction and hence may lack grounding).

Newell (1990) proposes an additional requirement for representations — namely that the application of encoding (sensing), internal transformations, and decoding (acting) must be executable on demand to the extent required to serve the purposes of the organism (which could be viewed essentially as the sensed internal environment of needs, drives, and emotions).

3.2 WHERE DO THE REPRESENTATIONS COME FROM?

Representations may be discovered by organisms (or evolution) by identifying the right medium of encoders (transducers) and decoders (effectors) and the right dynamics for the transformations for specific tasks. This would lead to a large number of task-specific *analogical* representations. Indeed, strong evidence for such analogical representations are can be found in living brains: the retinotopic maps in the visual cortex and the somatotopic maps of the sensory-motor cortex provide good examples of analogical representations (Kandell and Schwartz, 1985).

Alternatively, or in addition, a set of encoders and decoders may be used in conjunction with the ability to compose whatever sequence of transformations that may be necessary to form a representation. Most SAI systems take this route to the design of representations — by using a sufficiently general language (e.g., LISP) that allows the composition of whatever functions that may be necessary to satisfy the appropriate representation laws. Most NANN systems take the same route as well — they just happen to use a different language with a different set of primitives for composing the necessary transformations.

Irrespective of the approach chosen, the discovery of adequately powerful, efficient, and robust representations for any non-trivial set of tasks is still a largely unsolved problem. This is where learning and evolutionary processes play a major role. They must build the representations that perception and cognition utilize. One of the most informative characterizations of learning to date is in terms of storage of results of inference in a form that is suitable for use in the future (Michalski, 1993). Learning can clearly provide an avenue for the discovery of the necessary compositions of transformations which is a major aspect of representation. However, note that both SAI and NANN systems presuppose the existence of some representation before they can discover other useful representations. (Therefore it appears that representations cannot come into existence without the existence of physical transducers and effectors that connect such systems with the physical world, leading to the *grounding problem* — see below). This makes the initial representation or encoding extremely important. If it is not properly chosen (by the designer of the system or by evolution), it places additional (and perhaps insurmountable) burdens on the learning mechanisms (e.g., if the initial representation failed to capture spatial or temporal relations at a level of detail that is necessary for dealing with the problem domain).

It is far from clear that every task-specific representation ever used by the system must be learned. Representations may be constructed as necessary to solve specific problems and then discarded. Alternatively, some basic (learned or evolved) representations may be adapted in real time for solving specific problems. This is an important aspect of the schema-based approach to modelling intelligence proposed by Arbib (1994).

As already pointed out, living brains appear to provide a rich panoply of representations — including analogical and iconic representations in the form of serial-parallel networks of topography-preserving projections (Kuffler, Nicholls, and Martin, 1984; Zeki and Shipp, 1988; Uhr, 1986; Honavar, 1989; Honavar and Uhr, 1989a; 1989b) in the visual, auditory and motor cortices. Such representations have been largely ignored in today's SAI as well as NANN models. They may very well be among the essential representations grounded in the environment that form the foundation of a much larger representational edifice that is needed for human-like general intelligence.

In short, SAI and NANN systems often differ in terms of the preferred form of knowledge representation used although any knowledge that can be represented in one can also be represented (albeit not as efficiently, robustly or elegantly) in the other. The challenge for engineers and cognitive modellers is to choose the right mix of SAI and NANN (and whatever other possibilities that exist) to meet the needs of the problem at hand.

4 A CLOSER LOOK AT SAI AND NANN

Given the shared philosophical and scientific roots that SAI and NANN have in common, why the great fuss about their integration? Answering this question entails taking a closer look at some prototypical SAI and NANN systems followed by a critical examination of what are generally considered their defining characteristics and much-touted advantages and disadvantages. This examination demonstrates that despite assertions by some to the contrary, the differences between them are less than what they might seem at first glance; and to the extent they differ, such differences are far from being in any reasonable sense of the term, fundamental; and that the purported weaknesses of each can potentially be overcome through a judicious integration of techniques and tools selected from the other (Honavar, 1990; Honavar and Uhr, 1990a; Uhr and Honavar, 1994; Honavar and Uhr, 1994; Uhr, 1990; Boden, 1994).

4.1 PROBLEM SOLVING AS STATE SPACE SEARCH

State Space Search in SAI Systems

The prototypical SAI models are more or less direct descendents of the von Neumann stored program model of computation. The essential components of such a model are: a storage for programs (instructions for processing data), a processor for interpretation and execution of the program; and a (transient) working memory for receiving inputs, and holding intermediate results of processing. Learning programs have additional mechanisms for self-modification (i.e., the modification of the set of programs that they use).

The dominant paradigm for problem solving in SAI is *state space search* (Winston, 1992; Ginsberg, 1993). States represent snap-shots of the problem at various stages of its solution. Operators enable transforming one state into another. Typically, the states are represented using structures of symbols (e.g., lists). Operators transform one symbol structure (e.g., list, or a set of logical expressions) into another. The system's task is to find a path between two specified states in the state-space (e.g., the initial state and a specified goal, the puzzle and its solution, the axioms and a theorem to be proved, etc.).

In almost any non-trivial problem, a blind exhaustive search for a path will be impossibly slow, and there will be no known algorithm or a procedure for directly computing that path without resorting to search. As a result, much work in SAI has focused on the study of effective heuristic search procedures (Pearl, 1984). For example, SAI systems handle games like chess as follows: The initial board is established as the given, and a procedure is coded to compute whether a win-state has been reached. In addition, procedures are coded to execute legal moves and (usually) to compute heuristic assessments of the promise of each possible move, and to combine the separate heuristic assessments into an overall value that will be used to choose the next move. Finally, all these are put into a total structure that applies the appropriate heuristics, combines their results and evaluates alternative moves, and actually makes a move, then waits for and senses the opponent's moves, uses it to update the board (probably checking that it is indeed a legal move), and loops back to make its own next move. (For simplicity the look-ahead with minimax that most game-players use has been ignored, but that is essentially more of the same.)

Knowledge-Guided Search

Search in general can be guided by the knowledge that is at the disposal of the problem solver. If the system is highly specialized, the necessary knowledge is usually built into the search procedure (in the form of criteria for choosing among alternative paths, heuristic functions to be used, etc.). However, general purpose problem solvers also need to be able to retrieve problem-specific and perhaps even situation-specific knowledge to be used to guide the search during problem-solving. Indeed, such retrieval might itself entail search (albeit in a different space). Efficient, and flexible representations of such knowledge as well as mechanisms for their retrieval as needed during problem solving are, (although typically overlooked because most current AI systems are designed for very specialized, narrowly defined tasks), extremely important. This is an area where NANN or other implementations of content addressed memories and indexing schemes are especially worth exploring.

State Space Search in NANN Systems

The NANN system (a network of relatively simple processing elements, neurons, or nodes) is typically presented with an input pattern or initialized in a given starting state encoded in the form of a state vector each of whose elements corresponds to the state of a neuron in the network). It is designed or trained to output the correct response to each input pattern it receives (perhaps after undergoing a series of state updates determined by the rules governing its dynamic behavior). The input-output behavior of the network is a function of the network architecture, the functions computed by the individual nodes and parameters such as the weights.

For example, the solution of an optimization problem (traditionally solved using search) can be formulated as a problem of arriving at a state of a suitably designed network that corresponds to one of minimum energy (which is defined to correspond in some natural way to the optimality of the solution being sought). For an example of such an approach to theorem-proving, see (Pinkas, 1994). Ideally, the network dynamics are set up so as to accomplish this without additional explicit control. However, in practice, state updates in NANN systems are often controlled in a manner that is not much different from explicit control (as in sequential update of neurons in Hopfield networks (Hopfield, 1982) where only one neuron is allowed to change its state on any update cycle) to guarantee certain desired emergent behaviors). Indeed, a range of cognitive tasks do require selective processing of information that often necessitates the use of a variety of (albeit flexible and distributed) networks of controls that is presently lacking in most NANN models (Honavar and Uhr, 1990b). Many such control structures and processes are suggested by an examination of computers, brains, immune systems, and evolutionary processes.

In short, in both SAI and NANN systems, problem-solving involves state-space search; and although most current implementations tend to fall at one end of the spectrum or the other, it should be clear that there exists a space of designs that can use a mix of different state representations and processing methods. The choice of a particular design for a particular class of problems should primarily be governed by performance, cost, and reliability considerations for artificial intelligence applications and psychological and neurobiological plausibility for cognitive modelling.

4.2 Symbols, Symbol Structures, Symbolic Processes

Symbols

Knowledge representation as described earlier, generally implies the use of symbols at some level. The standard notion of a symbol is that it *stands for* something and when a symbol token appears within a symbolic expression carries the interpretation that the symbol stands for something within the context that is specified by its place in the expression. In general, a symbol serves as a surrogate for a body of knowledge that may need to be accessed and used in processing the symbol. And ultimately, this knowledge includes semantics or meaning of the symbol in the context in which it appears, including that provided by the direct or indirect grounding of the symbol structure in the external environment (Harnad, 1990).

Symbolic Processes

Symbolic processes are essentially transformations that operate on symbol structures to produce other symbol structures. Memory holds symbol structures that contain symbol tokens that can be modified by such processes. This memory can take several forms based on the time scales at which such modifications are allowed. Some symbol structures might have the property of determining choice and the order of application of transformations to be applied on other symbol structures. These are essentially the programs. Programs when executed — typically through the conventional process of compilation and interpretation and eventually — when they operate on symbols that are linked through grounding to particular effectors — produce behavior. Working memory holds symbol structures as they are being processed. Long-term memory, generally speaking, is the repository of programs and can be changed by addition, deletion, or modification of symbol structures that it holds.

Such a system can compute any Turing-computable function provided it has sufficiently large memory and its primitive set of transformations are adequate for the composition of arbitrarily symbol structures (programs) and the interpreter is capable of interpreting any possible symbol structure. This also means that any particular set of symbolic processes can be carried out by an NANN — provided it has potentially infinite memory, or finds a way to use its transducers and effectors to use the external physical environment to serve as its memory).

Knowledge in SAI systems is typically embedded in complex symbol structures such as lists (Norvig, 1992), logical databases (Genesereth and Nilsson, 1987), semantic networks (Quillian, 1968), frames (Minsky, 1975), schemas (Arbib, 1972; 1994), and manipulated by (often serial) procedures or inferences (e.g., list processing, application of production rules (Waterman, 1985), or execution of logic programs (Kowalski, 1977) carried out by a central processor that accesses and changes data in memory using addresses and indices.

It is often claimed that the NANN systems predominantly perform numeric processing in contrast to SAI systems which manipulate symbol structures.

Symbolic Processes in NANN systems

As already pointed out, NANN systems represent problem states using (typically binary) state vectors which are manipulated in a network of processors using (typically) numeric operations (e.g., weighted sums and thresholds). It is not hard to see that the numeric state vectors and transformations employed in such networks play an essential symbolic role although the rules of transformation may now be an emergent property of a large number of nodes acting in concert. In short, the formal equivalence of the various computational models guarantees that NANN can support arbitrary symbolic processes. It is not therefore surprising that several alternative mechanisms for variable binding and logical reasoning using NANN have been discovered in recent years. Some of these require explicit use of symbols (Shastri and Ajjanagadde, 1989); others resort to quasi-symbols that have some properties of symbols while not being actually symbols in their true sense (Pollack, 1990; Maclennan, 1994); still others use pattern vectors to encode symbols (Dolan and Smolensky, 1989; Smolensky, 1990; Sun, 1994a; Chen and Honavar, 1994). The latter approach to symbol processing is often said to use sub-symbolic encoding of a symbol as a pattern vectors each of whose components is insufficient in and of itself to identify the symbol in question (see the discussion on distributed representations below). In any case, most, if not all, of these proposals are implemented and simulated on general purpose digital computers, so none of the functions that they compute are outside the Turing framework.

4.3 NUMERIC PROCESSING

Numeric processing, as the name suggests, involves computations with numbers. On the surface it appears that most NANN perform essentially numeric processing. After all, the formal neuron of McCulloch and Pitts computes weighted sum of its numeric inputs. And the *neurons* in most NANN models perform similar numerical computations. On the other hand, SAI systems predominantly compute functions over structures of symbols. But numbers are in fact symbols for quantities; and any computable function over numbers can be computed by symbolic processes. In fact, general purpose digital computers have been performing both symbolic as well as numeric processing ever since they were invented.

4.4 Analog Processing

It is often claimed that NANN perform *analog computation*. Analog computation generally implies the use of dynamical systems describable using continuous differential equations. They operate in continuous time, generally with physical entities such as voltages, currents, which serve as physical *analogs* of the quantities of interest. Thus soap bubbles, servomechanisms, and cell membranes can all be regarded as analog computers (Rajaraman, 1981).

Whether physically realizable systems are truly analog or whether analog system is simply a mathematical idealization of (extremely fine-grained) discrete system is a question that borders on the philosophical (e.g., are matter, time and space continuous or discrete?). However, some things are fairly clear. Most NANN are simulated on digital computers and compute in discrete steps and hence are clearly not analog. The few NANN models can be regarded as analog devices — e.g., the analog VLSI circuits designed and built by Carver Mead and colleagues (Mead, 1989) — are incapable of discrete symbolic computations (because of their inability to make all-or-none or discrete choices) (Maclennan, 1994) although they can approximate such computations. (For example, the stable states or attractors of such systems can be interpreted as identifiable discrete states).

Analog systems can be, and often are simulated on digital computers at the desired level of precision. However, this might involve a time-consuming iterative calculation to produce a result that could potentially be obtained almost instantaneously (and transduced using appropriate transducers) given the right analog device. Thus analog processing appears to be potentially quite useful in many applications (especially those that involve perceptual and motor behavior). It is possible that evolution has equipped living systems with just the right repertoire of analog devices that help them process information in this fashion. However, it is somewhat misleading to call such processing *computation* (in the sense defined by Turing) because it lacks the discrete combinatorial structure that is characteristic of all Turing-equivalent models of computation (Maclennan, 1994).

Whether analog processes play a fundamental role (beyond being part of grounding of representations) in intelligent systems remains very much an open question. It is also worth pointing out that digital computers can, and in fact do, make use of essentially analog devices such as transistors but they use only a few discrete states to support computation (in other words, the actual analog value is irrelevant so long as it lies withing a range that is distinguishable from some other range). And when embedded in physical environments, both SAI and NANN systems do encounter analog processes through sensors and effectors.

4.5 COMPOSITIONALITY AND SYSTEMATICITY OF REPRESENTATION

It has been argued by many e.g., Fodor and Pylyshyn, 1988) that *compositionality* and *systematicity* (structure sensitivity) of representation are essential for explaining mind. In their view, NANN are inadequate models of mind because NANN representations lack these essential properties. Compositionality is the property that demands that representations must possess an internal *syntactic structure* as a consequence of a particular method for composing complex symbol structures from simpler components. Systematicity requires the existence of processes that are sensitive to the syntactic structure. As argued by Sharkey and Jackson (1994), lack of compositionality is demonstrably true only for a limited class of NANN representations; and compositionality and systematicity in and of themselves are inadequate to account for cognition (primarily for lack of grounding or semantics). Van Gelder and Port (1994) have shown that several forms of compositionality can be found in NANN representations.

4.6 GROUNDING AND SEMANTICS

Many in the artificial intelligence and cognitive science research community agree on the need for *grounding* of symbolic representations through sensory (e.g., visual, auditory, tactile) transducers and motor effectors in the external environment on the one hand and the internal environment of needs, drives, and emotions of the organism (or robot) in order for such representations (which are otherwise devoid of any intrinsic meaning to the organism or robot) to become imbued with meaning or semantics (Harnad, 1990). Some have argued that NANN systems provide the necessary apparatus for grounding (Harnad, Hanson, and Lubin, 1994). It is important to realize that NANN as computational models do not provide physical grounding (as opposed to grounding in a simulated world of virtual reality) for representations any more than their SAI counterparts. It is only the physical systems with their physical substrate on which the representations reside that are capable of providing such grounding in physical reality when equipped with the necessary transducers and effectors. This is true irrespective of whether the system in question is a prototypical SAI system, or a prototypical NANN system, or a hybrid or integrated system.

4.7 Serial versus Parallel Processing

As pointed out earlier, most of today's SAI systems are serial programs that are executed on serial von Neumann computers. However, serial symbol manipulation is more an artifact of most current implementations of SAI systems than a necessary property of SAI. In parallel and distributed computers, memory is often locally available to the processors and even can be almost eliminated in data flow machines which model functional or applicative programs where data is transformed as it flows through processors or functions. Search in SAI systems can be, and often is, parallelized by mapping the search algorithm onto a suitable network of computers (Uhr, 1984; 1987; Hewitt, 1977; Hillis, 1985) with varying degrees of centralized or distributed control. Many search problems that arise in applications such as temporal reasoning, resource allocation, scheduling, vision, language understanding and logic programming can be formulated as constraint satisfaction problems which often lend themselves to solution using a mix of serial and parallel processing (Tsang, 1993).

Similarly, SAI systems using production rules can be made parallel by enabling many rules to be matched simultaneously in a data flow fashion (as in RETE pattern matching networks (Forgy, 1982)). Multiple matched rules may be allowed to fire and change the working memory in parallel as in parallel production systems (Uhr, 1979) and classifier systems (Holland, 1975) — so long as whenever two or more rules demand conflicting actions, arbitration mechanisms are provided to choose among the alternatives or resolve such conflicts at the sensory-motor interface. Such arbitration mechanisms can themselves be realized using serial, parallel (e.g., winner-take-all mechanism), or serial-parallel (e.g., pyramid-like hierarchies of decision mechanisms) networks of processes.

NANN systems with their potential for massive fine-grained parallelism of computation offer a natural and attractive framework for the development of highly parallel architectures and algorithms for problem solving and inference. Such systems are considered necessary by many researchers (Uhr, 1980; Feldman and Ballard, 1982) for tasks such as real-time perception. But SAI systems doing symbolic inference can be, and often are, parallelized, and certain inherently sequential tasks need to be executed serially. On any given class of problems, the choice of decomposition of the computations to be performed into a parallel-serial network of processes and their mapping onto a particular network of processors has to be made taking the cost and performance tradeoffs into consideration.

4.8 KNOWLEDGE ENGINEERING VERSUS KNOWLEDGE ACQUISITION THROUGH LEARNING

The emphasis in some SAI systems (especially the so-called knowledge-based expert systems (Waterman, 1985)) on knowledge engineering has led some to claim that SAI systems are, unlike their NANN counterparts, incapable of learning from experience. This is clearly absurd as even a cursory look at the current research in machine learning (Shavlik and Dietterich, 1990; Buchanan and Wilkins, 1993) and much early work in pattern recognition (Uhr, 1973; Fu, 1982; Miclet, 1986) shows. Research in SAI and closely related systems indeed have provided a wide range of techniques for deductive (analytical) and inductive (synthetic) learning. Learning by acquisition and modification of symbol structures almost certainly plays a major role in knowledge acquisition in humans who learn and communicate in a wide variety of natural languages (e.g., English) as well as artificial ones (e.g., formal logic, programming languages). While NANN systems with their micro-modular architecture offer a range of interesting possibilities for learning, for the most part, only the simplest parameter or weight modification algorithms have been explored to date (McClelland and Rumelhart, 1986 et almr86; Kung, 1993; Haykin, 1993). In fact, learning by weight modification alone appears to be inadequate in and of itself to model rapid and irreversible learning that is observed in many animals. Algorithms that modify networks through structural changes that involve the recruitment of neurons (Greenough and Bailey, 1988; Honavar, 1989; 1990; Honavar and Uhr, 1989a; 1989b; 1993; Kung, 1993; Grossberg, 1980) appear promising in this regard.

A detailed discussion of learning is beyond the scope of this chapter. Suffices it to point out that most forms of learning can be understood and implemented in terms of structures and processes for representing and reasoning with knowledge (broadly interpreted) and for memorizing the results of such inference in a form that lends itself to retrieval and use at a later time (Michalski, 1993). Thus any NANN or SAI or some hybrid architecture that is capable of performing inference and has memory for storing the results of inference for retrieval and use on demand can be equipped with the ability to learn. The interested reader is referred to (Honavar, 1994) for a detailed discussion of systems that learn using multiple strategies and representations. Additional examples of systems that combine NANN and SAI approaches to learning can be found in (Uhr, 1973; Holland, 1975; Honavar, 1992; 1994; Honavar and Uhr, 1993; Lacher and Nguyen, 1994; Carpenter and Grossberg, 1994; Shavlik, 1994; Goldfarb and Nigam, 1994; Booker, Riolo, and Holland, 1994). In short, SAI systems offer powerful mechanisms for manipulation of highly expressive structured symbolic representations while NANN offer the potential for robustness, and the ability to fine-tune their use as a function of experience (primarily due to the use of tunable numeric weights and statistics).

4.9 Associative as Opposed to Address-Based Storage and Recall

An often cited distinction between SAI and NANN systems is that the latter employ associative (i.e., content-addressable) as opposed to the address-and-index based storage and recall of patterns in memory typically used by the former. This is a misconception for several reasons: Address-and-index based memory storage and retrieval can be used to simulate content-addressable memory and vice versa and therefore unless one had access to the detailed internal design operation of such systems, their behavior can be indistinguishable from each other. Many SAI systems conventional computers use associative memories in some form or another (e.g., hierarchical cache memories). While associative recall may be better for certain tasks, address (or location-based) recall may be more appropriate for others. Indeed, many computational problems that arise in symbolic inference (pattern matching and unification in rule-based production systems or logic programming) can take advantage of associative memories for efficient processing (Chen and Honavar, 1994).

In prototypical NANN models, associative recall is based on some relatively simple measure of proximity or closeness (usually measured by Hamming distance in the case of binary patterns) to the stored patterns. While this may be appropriate in domains in which related items have patterns or codes that are close to each other, it would be absurd to blindly employ such a simple content-addressed memory model in domains where symbols are arbitrarily coded for storage (which would make hamming distance or a similar proximity measure useless in recalling the associations that are really of interest). Establishing (possibly context-sensitive) associations between otherwise arbitrary symbol structures based on their meanings and retrieving such associations efficiently requires complex networks of learned associations more reminiscent of associative knowledge networks, semantic networks (Quillian, 1968), frames (Minsky, 1975), conceptual structures (Sowa, 1984), schemas (Arbib, 1994), agents (Minsky, 1986) and object-oriented programs of SAI (Norvig, 1992) than today's simple NANN associative memory models. This is not to suggest that such structures cannot be implemented using suitable NANN building blocks — see (Arbib, 1994; Dyer, 1994; 1994b; Miikkulainen, 1994a; Bookman, 1994; Barnden, 1994a; 1994b) for some examples of such implementations. Indeed, such NANN implementations of complex symbol structures and symbolic processes can offer many potential advantages (e.g., robustness, parallelism) for SAI.

4.10 DISTRIBUTED STORAGE, PROCESSING, AND CONTROL

Distributed storage, processing, and control are often claimed to be some of the major advantages of NANN systems over their SAI counterparts. It is far from clear as to what is generally meant by the term *distributed* when used in this context (Oden, 1994).

Perhaps it is most natural to think of an item as distributed when it is coded (say as a pattern vector) whose components by themselves are neither sufficient to identify the item nor have any useful semantic content. Thus, the binary code for a letter of the alphabet is distributed. Any item thus distributed eventually has to be reconstructed from the pieces of its code. This form of distribution may be in space, time, or both. Thus the binary code for a letter of the alphabet may be transmitted serially (distributed in time) over a single link that can carry 1 bit of information at a time or in parallel (distributed in space) using a multi-wire bus. If a system employs such a mechanism for transmission or storage of data, it also needs decoding mechanisms for reconstructing the coded item at the time of retrieval. It is easy to see that this is not a defining property of NANN systems as it is found in even the serial von Neumann computers. In any event, both NANN as well as SAI systems can use such distributed coding of symbols. And, as pointed out by Hanson and Burr (1990), distributed coding in and of itself, offers no representational capabilities that are not realizable using a non-distributed coding.

In the context of NANN, the term *distributed* is often used to refer to storage of parts of an item in a unit where parts of other items also stored (for example, by superposition). Thus, each unit participates in storage of multiple items and each item is distributed over multiple units. (There is something disconcerting about this particular use of the term distributed in a technical sense: Clearly, one can invent a new name for whatever it is that a unit stores — e.g., a number whose binary representation has a '1' in its second place. Does the system cease to be distributed as a result?). It is not hard to imagine an analogous notion of distribution in time instead of space but it is also fraught with similar semantic difficulty.

The term *distributed* when used in the context of *parallel and distributed processing*, generally refers to the decomposition of a computational task into more or less independent pieces that are executed on different processors with little or no inter-processor communication (Uhr, 1984; 1987; Almasi and Gottlieb, 1989). Thus many processors may perform the same computation on pieces of the data (as in single-instruction-multiple data or SIMD computer architectures) or each processor may perform a different computation on the same data e.g., computation of various intrinsic properties of an image (as in multiple-instruction-single-data or MISD computer architectures), or a combination of both (as in multiple-instruction-multiple-data or MIMD computer architectures). Clearly, both NANN and SAI systems can take advantage of such parallel and distributed processing. The reader is referred to (Almasi and Gottlieb, 1989; Uhr, 1984; 1987) for examples.

4.11 REDUNDANCY AND FAULT TOLERANCE

Often the term *distributed* is used more or less synonymously with *redundant* and hence fault-tolerant in the NANN literature. This is misleading because there are many ways to ensure redundancy of representation, processing and control. One of the simplest involves storing multiple copies of items and/or using multiple processors to replicate the same computation in parallel, and using a simple majority vote or more sophisticated statistical evidence combination processes to pick the result. Redundancy and distributivity are orthogonal properties of representations. And clearly, SAI as well as NANN systems can be made redundant and fault-tolerant using the same techniques.

4.12 STATISTICAL, FUZZY, OR EVIDENTIAL INFERENCE

It is often claimed that NANN models provide noise-tolerant and robust inference because of the probabilistic, fuzzy, or evidential nature of the inference mechanisms used. This is largely due to combination and weighting of evidence from multiple sources through the use of numerical weights or probabilities. It is possible to establish the formal equivalence inference in certain classes of NANN models with probabilistic or fuzzy rules of reasoning. But fuzzy logic (Zadeh, 1975; Yager and Zadeh, 1993) operates (as its very name suggests), with *logical* (hence symbolic) representations. Probabilistic reasoning is an important and active area of research in SAI as well (See Pearl, 1988 for details). Heuristic evaluation functions that are widely used in many SAI systems provide additional examples of approximate, that is, not strictly truth-preserving inference in SAI systems. In many SAI systems, the requirements of soundness and completeness of inference procedures are often sacrificed in exchange for efficiency. In such cases, additional mechanisms are used to (after the fact) verify and if necessary, override the results of inference if they are found to conflict with other evidence.

Much research on human reasoning indicates that people occasionally draw inferences that are logically unsound (Johnson-Laird and Byrne, 1991). This suggests that although people may be capable of applying sound inference procedures, they probably take shortcuts when faced with limited computational or memory resources. Approximate reasoning under uncertainty is clearly an important tool that both SAI and NANN systems can potentially employ to effectively make rapid, usually reliable and useful, but occasionally fallible inferences in real time.

4.13 SAI AND NANN AS MODELS OF MINDS/BRAINS

Some of the SAI research draws its inspiration from (rather superficial) analogies with the mind and mental phenomena and in turn contributes hypotheses and models to the study of minds; Similarly, many NANN models draw their inspiration from (albeit superficial) analogies with the brain and neural phenomena and in turn contribute models that occasionally shed light on some aspects of brain function (Churchland and Sejnowski, 1992).

It is important to emphasize that neither today's SAI nor today's NANN have the monopoly on modelling minds and brains. Today's NANN models are at best, extremely simplified caricatures of biological neural networks (Shepherd, 1989; 1990; McKenna, 1994). Biological neurons and microcircuits of neurons provide computational primitives that are far more powerful than simple threshold or sigmoids that are used in most NANN models (Uhr, 1994). Brains display highly structured yet flexible organization into regions, layers, and modules that perform specialized functions (Kuffler, Nicholls and Martin, 1984; Zeki and Shipp, 1988). Such networks may be modelled by highly structured NANN models that organize the neurons into locally connected, topography preserving layers that are organized in loosely hierarchical fashion (Uhr, 1986; Honavar and Uhr, 1989a; 1989b; Honavar, 1992). Such structures appear to organize the networks of the brain in space (in ways that reflect the physics of the environment using networks of analog representations) and time (through the use of feedback loops with varying amounts of delay, networks of clocks and osciallators).

The brain appears to perform symbolic, numeric, as well as analog processing. The pulses transmitted by neurons are digital; the membrane voltages are analog (continuous); The molecular level phenomena that involve closing and opening of channels appears to be digital; The diffuse influence of neurotransmitters and hormones appear to be both analog and digital.

Changes in learning appear to involve both gradual changes of the sort modeled by the parameter changing or weight modification algorithms of todays NANN as well as major structural changes involving the recruitment of neurons and changes in network topology (Greenough and Bailey, 1988; Honavar, 1989; 1990; Honavar and Uhr, 1989a; 1989b; 1993). In fact, learning by weight modification alone appears to be inadequate in and of itself to model rapid and irreversible learning that is observed in many animals.

Also missing from most NANN models are elaborate control structures and processes of the sort found in brains including networks of oscillators that control timing. Perception, learning and control in brains appear to utilize events at multiple spatial and temporal scales (Grossberg, 1982). Additional processes not currently modelled by NANN systems include processes that include networks of markers that guide neural development, structures and processes that carry information that might be used to generate other network structures, and so on (Honavar and Uhr, 1990).

Clearly, living minds/brains are among the few examples of truly versatile intelligent systems that we have today. They are our existence proof that such systems are indeed possible. So even those whose primary interests are in constructing artificial intelligence systems can ill afford to ignore the insights offered by a study of biological intelligence (McKenna, 1994). (This does not of course mean that such an effort cannot exploit alternative technologies to accomplish the same functions, perhaps even better than their natural counterparts). But it is a misconception to assume that today's NANN model brains any more than today's SAI programs model minds. In short, the processes of the minds appear to be far less rigidly structured and far more flexible than today's SAI systems and the brains appear to have a lot more structure, organization, and control than today's homogeneous networks of simple processing elements which we call NANN. A rich space of designs that combine aspects of both within a well-designed architecture for intelligence remains to be explored.

5 INTEGRATION OF SAI AND NANN

It must be clear from the discussion in the previous sections that at least on the surface it looks like SAI and NANN are each appropriate, and possibly even necessary for certain problems, and grossly inappropriate, almost impossible, for others. But of course each can do anything that the other can. The issues are ones of performance, efficiency and elegance (and in cognitive modelling, perhaps plausibility in terms of the various known constraints between different levels — such as psychological, neurobiological, and neurochemical — at which satisfactory explanations are sought), and not theoretical capabilities as computational models.

This is a common problem in computing. One computer or programming language may be extremely well-suited for some problems but awful for others, while a second computer or language may be the opposite. This suggests several engineering possibilities (Uhr and Honavar, 1994), including:

1. Try to re-formulate and re-code the problem to better fit the computer or language.
2. Use one computer or language for some parts of the process and the other for others.
3. Build a new computer or language that contains constructs from each, and use these as appropriate.
4. Try to find as elegant as possible a set of primitives that underlie both computers or languages, and use these to build a new system.

The term *hybrid* is beginning to be used for systems that in some way try to combine SAI and NANN. If any of the above is called a hybrid probably all of the others should also. But usually hybrid refers to systems of type [2] or [3]. Types [3] and [4] would appear to be better than [2] (although harder to realize), since they would probably be more efficient and more elegant. Thus the capabilities of both SAI and NANN should be combined by tearing them apart to the essential components of their underlying processes and integrating these as closely as possible. Then the problem should be re-formulated and re-coded to fit this new system as well as possible. This restates a general principle most people are coming to agree on with respect to the design of multi-computer networks and parallel and distributed algorithms: the algorithm and the architecture should be designed to fit together as well as possible, giving algorithm-structured architectures and architecture-structured algorithms (Uhr, 1984; 1987; Almasi and Gottlieb, 1989).

6 SUMMARY

SAI and NANN each demonstrate at least one way of performing certain tasks naturally and thus pose the interesting problem for the other of doing something equivalent perhaps more elegantly, efficiently, or robustly than the other. It should be clear from the discussion above that the integration of SAI and NANN systems can be beneficially explored along several dimensions.

In the short term, *hybrid* architectures that use NANN and SAI modules to perform different but well-coordinated sets of functions in specific applications are definitely worth exploring. A partial list of examples of such integration include: neural network and expert knowledge based systems (Lin and Hendler, 1994; Shavlik, 1994; Gallant, 1993; Medsker, 1994); systems for language processing (Bookman, 1994; Dyer, 1994a; 1994b; Miikkulainen, 1994a; 1994b; Barnden, 1994b; Omlin and Giles, 1994; Smolensky, Legendre, and Miyata, 1994; Servan-Schreiber, Cleeremans, and McClelland, 1994); systems for visual pattern recognition and spatial reasoning (Honavar and Uhr, 1989a; 1989b; Honavar and Uhr, 1994; Honavar, 1994; Ballard and Brown, 1982; Uhr, 1987; Tanimoto and Klinger, 1980; Wechsler, 1990; Duda and Hart, 1973; Fu, 1982; Miclet, 1982; Carpenter and Grossberg, 1994; Kosslyn and Jacobs, 1994; Mjolsness, 1994); systems for symbolic inference (Sun, 1994a; 1994b; Barnden, 1994b; Smolensky, 1990; Shastri and Ajjanagadde, 1989; Chen and Honavar, 1994); systems for learning (Honavar and Uhr, 1989a; 1989b; 1993; Honavar, 1992; 1994; Shavlik, 1994; Goldfarb and Nigam, 1994; Fu, 1982;

Fukunaga, 1990; Gallant, 1994; Uhr, 1973; Holland, 1975; Booker, Riolo, and Holland, 1994; Lacher and Nguyen, 1994; Carpenter and Grossberg, 1994; Dyer, 1994a). These efforts offer a number of important insights into the design and performance of such hybrid systems for cognitive modelling on the one hand and engineering intelligent systems for practical applications on the other (see below).

The integration of concepts, constructs, techniques and technologies drawn from SAI and NANN as well as other closely related paradigms (including statistical pattern recognition, syntactic pattern recognition, evolutionary computation) offers a rich and potentially very promising design space for exploration by artificial intelligence engineers and cognitive theorists. It is becoming increasingly obvious that this space exhibits almost infinite variety that is characteristic of complex systems. In the long-term, a coherent theoretical framework for analysis and synthesis of such systems has to be developed. In order to do this, we need to start developing and refining our categorization of such mutually inter-related systems. One way to approach this task is to seek categorizations that capture essential underlying principles of the *architecture*, alternative *implementations* of the architecture, and finally alternative or *physical realization* of the candidate implementations of such systems relative to our goals of understanding and engineering intelligence.

Because of the engineering and technological emphasis of artificial intelligence, most research in the area has focused on the development of algorithms for specific tasks that appear to require intelligence if performed by humans (e.g., diagnosis, planning, character recognition). While such efforts provide useful technological tools in the short term, they appear to have fallen short of providing much insight into alternative implementations and physical realizations of architectures for general intelligence.

Most artificial intelligence and cognitive science theories of intelligence are primarily about the content of knowledge or types of knowledge for some task of interest, with minimal commitment on the choice of architecture (or equivalently, the programming language that defines the virtual architecture of the computer). Perhaps this is because it is tacitly assumed that any such architecture is one that is capable of supporting universal computation and that nothing else about it is of much interest. Perhaps this is where the dichotomy between SAI and NANN can help focus our attention on architectural issues. After all, NANN models do (in most cases) represent architectural commitment(s) that are different from those implicitly assumed by SAI models (e.g., lambda calculus or production systems). However, it must be emphasized that one architectural commitment is not necessarily better than another independent of the task for which the architecture is used. Also worth noting is the fact that the same system may be lend itself to multiple architectural descriptions. Each such description can potentially add to our understanding of different aspects of the system in important ways. Furthermore, each architectural description lends itself to multiple implementations; For example, the same architecture can be implemented using a network of simple processors or simulated by a program on a conventional serial computer. And each implementation lends itself to multiple physical realizations.

Living minds/brains offer an existence proof of at least one architecture for general intelligence. SAI and NANN paradigms together offer a wide range of architectural choices. Each architectural choice brings with it some obvious (and some not so obvious) advantages as well as disadvantages in the solution of specific problems using specific algorithms, given certain performance demands and design constraints imposed by the available choices of physical realizations of the architecture. Together, the cross-product of the space of architectures, algorithms, and physical realizations constitutes a large and interesting space of possible designs for intelligent systems. Examples of systems resulting from a judicious integration of concepts, constructs, techniques and technologies drawn from both traditional artificial intelligence systems and artificial neural networks clearly demonstrate the potential benefits of exploring this space. And, perhaps more importantly, the rather severe practical limitations of today's SAI and NANN systems strongly argues for the need for a systematic exploration of such design space.

This suggests that it might be fruitful to approach the choice of architectures, implememtations, and their physical realizations using the entire armamentarium of tools drawn from the theory and practice of computer science — including the design of programming languages (and hence virtual architectures), computers, algorithms, and programs. Our primary task is to identify subsets of Turing-computable functions necessary for general intelligence, an appropriate mix of architectures for supporting specific subsets of these functions, as well as appropriate realizations of such architectures in physical devices. The hybrid or integrated SAI-NANN designs explored to date — including those examined in several recent books on this subject (Honavar and Uhr, 1994a; Sun and Bookman, 1994; Goonatilake and Khebbal, 1994; Levine and Aparicioiv, 1994) are only suggestive of a much larger space of interesting possibilities. It is almost certainly premature to pick one architecture over another as the architecture of choice for general intelligence (of the sort attributed to humans), or even eliminate certain architectures from consideration as candidates. Such choices can be made only after a careful evaluation of possible designs.

ACKNOWLEDGEMENTS

This work was partially supported by the Iowa State University College of Liberal Arts and Sciences and the National Science Foundation grant IRI-9409580. The author is indebted to Professors Leonard Uhr and Gregg Oden for extensive discussions on the topic of this chapter. He would like to thank Dr. Ron Sun and Dr. Larry Bookman for their invitation to him to contribute to this book as well as their comments on an earlier draft of this chapter.

REFERENCES

[1] Almasi, G.S. and Gottlieb, A. (1989). *Highly Parallel Computing.* New York: Benjamin-Cummings.

[2] Arbib, M.A. (1972). *The Metaphorical Brain.* New York: Wiley-Interscience.

[3] Arbib, M.A. (1994). Schema Theory: Cooperative Computation for Brain Theory and Distributed AI. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[4] Ballard, D. and Brown, C. (1982) *Computer Vision.* Englewood Cliffs, NJ: Prentice Hall.

[5] Barnden, J.A. (1994a). How Might Connectionist Networks Represent Propositional Attitudes? In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[6] Barnden, J.A. (1994b). Complex Symbol Processing in a Transiently Localist Connectionist Architecture. In: *Computational Architectures Integrating Symbolic and Neural Processes.* Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[7] Bickhard, M.H. (1993). Troubles With Computationalism. (draft)

[8] Boden, M. (1994). Horses of a Different Color? In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[9] Booker, L.E., Riolo, R.L., and Holland, J.H. (1994). Learning and Representation in Classifier Systems. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[10] Bookman, L.A. (1994). A Framework for Integrating Relational and Associational Knowledge for Comprehension. In: *Computational Architectures Integrating Symbolic and Neural Processes.* Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[11] Buchanan, B.G. and Wilkins, D.C. (1993). *Readings in Knowledge Acquisition and Learning.* San Mateo, CA: Morgan Kaufmann.

[12] Carpenter, G. and Grossberg, S. (1994). Integrating Symbolic and Neural Processes in a Self-Organizing Architecture for Pattern Recognition and Prediction. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[13] Chandrasekaran, B. and Josephson, S.G. (1994). Architecture of Intelligence: The Problems and Current Approaches to Solutions. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[14] Chen, C. and Honavar, V. (1994). Neural Networks for Inference. In preparation.

[15] Churchland, P.S. (1986). *Neurophilosophy.* Cambridge, MA: MIT Press.

[16] Churchland, P.S. and Sejnowski, T.J. (1992). *The Computational Brain.* Boston, MA: MIT Press.

[17] Cohen, D. (1986). *Introduction to Computer Theory.* New Tork: Wiley.

[18] Cooper, E.D. (1990). An object-oriented interpreter for symbol train processing. (Preprint).

[19] Dolan, C.P. and Smolensky, P. (1989). Tensor product production system: A modular architecture and representation. *Connection Science,* 1:53-58.

[20] Duda, R.O. and Hart, P.E. (1973). *Pattern Classification and Scene Analysis.* New York: Wiley.

[21] Dyer, M. (1994). Grounding Language in Perception. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[22] Feigenbaum, E.A. (1963). In: *Computers and Thought.* Feigenbaum, E.A. and Feldman, J. (Ed.) New York: McGraw-Hill.

[23] Feldman, J.A. and Ballard, D.H. (1982). Connectionist models and their properties. *Cognitive Science,* 6:205-264.

[24] Fodor, J. (1976). *The Language of Thought.* Boston, MA: Harvard University Press.

[25] Fodor, J. and Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis. In: *Connections and Symbols.* Pinker, S. and Mehler, J. (Ed.) Cambridge, MA: MIT Press.

[26] Forgy, C.L. (1982). RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence,* 19:17-37.

[27] Fu, K.S. (1982). *Syntactic Pattern Recognition and Applications.* Englewood Cliffs, NJ: Prentice Hall.

[28] Fukunaga, K. (1990). *Introduction to Statistical Pattern Recognition.* New York: Academic Press.

[29] Gallant, S. (1993). *Neural Networks and Expert Systems.* Cambridge, MA: MIT Press.

[30] Genesereth, M.R., and Nilsson, N.J. (1987). *Logical Foundations of Artificial Intelligence.* Palo Alto, CA: Morgan Kaufmann.

[31] Ginsberg, M. (1993). *Essentials of Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

[32] Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Reading, MA: Addison-Wesley.

[33] Goldfarb, L. and Nigam, S. (1994). The Unified Learning Paradigm: A Foundation for AI. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[34] Goonatilake, S. and Khebbal, S. (1994). (Ed.) *Hybrid Intelligent Systems.* London: Wiley.

[35] Greenough, W.T. and Bailey, C.H. (1988). The Anatomy of Memory: Convergence of Results Across a Diversity of Tests. *Trends in Neuroscience* 11, 142-147.

[36] Grossberg, S. (1982). *Studies of Mind and Brain.* Boston, MA: Reidel.

[37] Hanson, S.J. and Burr, D. (1990). What Connectionist Models Learn: Learning and Representation in Connectionist Networks. *Behavior and Brain Sciences*, 13:1-54.

[38] Harnad, S. (1990). The Symbol Grounding Problem. *Physica D*, 42:335-346.

[39] Harnad, S., Hanson, S.J., and Lubin, J. (1994). Learned Categorical Perception in Neural Nets: Implications for Symbol Grounding. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[40] Haykin, S. (1994). *Neural Networks.* New York: Macmillan.

[41] Hewitt, C. (1977). Viewing Control Structures as Patterns of Passing Messages. *Artificial Intelligence*, 8:232-364.

[42] Hillis, D. (1985). *The Connection Machine.* Cambridge, MA: MIT Press.

[43] Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

[44] Hopfield, J.J. (1982). Neural Networks and Physical Systems With Emergent Collective Computational Abilities. *Proceedings of the National Academy of Sciences*, 79:2554-2558.

[45] Honavar, V. (1989). Perceptual Development and Learning: From Behavioral, Neurophysiological and Morphological Evidence to Computational Models. Tech. Rep. 818. Computer Sciences Dept., University of Wisconsin, Madison, Wisconsin.

[46] Honavar, V. (1990). *Generative Learning Structures for Generalized Connectionist Networks.* Ph.D. Dissertation. University of Wisconsin, Madison, Wisconsin.

[47] Honavar, V. (1992). Inductive Learning Using Generalized Distance Measures. In: *Proceedings of the SPIE Conference on Adaptive and Learning Systems.* Orlando, Florida.

[48] Honavar, V. (1994). Toward Learning Systems That Use Multiple Strategies and Representations. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[49] Honavar, V. and Uhr, L. (1989a). Brain-Structured Connectionist Networks that Perceive and Learn. *Connection Science*, 1:139-159.

[50] Honavar, V. and Uhr, L. (1989b). Generation, Local Receptive Fields, and Global Convergence Improve Perceptual Learning in Connectionist Networks. In: *Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, San Mateo, CA: Morgan Kaufmann.

[51] Honavar, V. and Uhr, L. (1990a). Symbol Processing Systems, Connectionist Networks, and Generalized Connectionist Networks. Tech. Rep. 90-23. Department of Computer Science, Iowa State University, Ames, Iowa.

[52] Honavar, V. and Uhr, L. (1990b). Coordination and Control Structures and Processes: Possibilities for Connectionist Networks. *Journal of Experimental and Theoretical Artificial Intelligence*, 2:277-302.

[53] Honavar, V. and Uhr, L. (1993) Generative Learning Structures and Processes for Generalized Connectionist Networks, *Information Sciences*, 70:75-108.

[54] Honavar, V. and Uhr, L. (1994a). (Ed.) *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* San Diego, CA.: Academic Press.

[55] Honavar, V. and Uhr, L. (1994b). Toward Integrated Architectures for Artificial Intelligence and Cognitive Modelling. In: *Intelligent Hybrid Systems.* Goonatilake, S. and Khebbal, S. (Ed). London: Wiley.

[56] Johnson-Laird, P. and Byrne, J. (1991). *Deduction.* New York: Lawrence Erlbaum.

[57] Klir, G.J. (1969). *An Approach to General Systems Theory.* New York: Van Nostrand Reinhold.

[58] Klir, G.J. (1985). *Architecture of Systems Problem Solving.* New York: Plenum.

[59] Kosslyn, S. and Jacobs, R.A. (1994). Encoding Shape and Spatial Relations: A Simple Mechanism for Coordinating Multiple Representations. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[60] Kowalski, R.A. (1977). *Predicate Logic as a Programming Language.* Amsterdam: North-Holland.

[61] Koza, J. (1992). *Genetic Programming.* Boston, MA: MIT Press.

[62] Kuffler, S.W., Nicholls, J.G., and Martin, A.R. (1984). *From Neuron to Brain.* Sunderland, MA: Sinaur.

[63] Kung, S. Y. (1993). *Digital Neural Networks.* New York: Prentice Hall.

[64] Lacher, R.C. and Nguyen, K.D. (1994). Hierarchical Architectures for Reasoning. In: *Computational Architectures Integrating Symbolic and Neural Processes.* Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[65] Levine, D.S. and Aparicioiv, M. (1994). (Ed.) *Neural Networks for Knowledge Representation.* New York: Lawrence Erlbaum.

[66] Lin, C. and Hendler, J. (1994). A Study of a Hybrid Connectionist-Symbolic System for The Analysis of Ballistic Signals. In: *Computational Architectures Integrating Symbolic and Neural Processes.* Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[67] Maclennan, B.J. (1994). Image and Symbol — Continuous Computation and the Emergence of the Discrete. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[68] McCulloch, W.S. and Pitts, W. (1943). A Logical Calculus of the Ideas Immanent in Neural Activity. *Bulletin of Mathematical Biophysics,* 5:115-137.

[69] McClelland J., Rumelhart, D. and the PDP Research Group (1986). (Ed.) *Parallel Distributed Processing.* Boston, MA: MIT Press.

[70] McKenna, T. (1994). The Role of Inter-Disciplinary Research Involving Neuroscience in the Design of Intelligent Systems. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[71] Mead, C. (1989). *Analog VLSI and Neural Systems.* Reading, MA: Addison-Wesley.

[72] Medsker, L. (1994). *Hybrid Neural Network and Expert Systems.* Boston: Kluwer.

[73] Miikkulainen, R. (1994a). Integrated Connectionist Models: Building AI Systems on Subsymbolic Foundations. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[74] Miikkulainen, R. (1994b). Subsymbolic Parsing of Embedded Structures. In: *Computational Architectures Integrating Symbolic and Neural Processes*. Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[75] Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs*. New York: Springer-Verlag.

[76] Michalski, R.S. (1993). Toward a Unified Theory of Learning: Multi-Strategy Task-Adaptive Learning. In: *Readings in Knowledge Acquisition and Learning*. Buchanan, B.G., and Wilkins, D.C. (Ed.) San Mateo, CA: Morgan Kaufmann.

[77] Miclet, L. (1986). *Structural Methods in Pattern Recognition*. New York: Springer-Verlag.

[78] Minsky, M. (1963). Steps Toward Artificial Intelligence. In: *Computers and Thought*. Feigenbaum, E.A. and Feldman, J. (Ed.) New York: McGraw-Hill.

[79] Minsky, M. (1975). A Framework for Representing Knowledge. In: *The Psychology of Computer Vision*. Winston, P. H. (Ed). New York: McGraw-Hill.

[80] Minsky, M. (1986). *Society of Mind*. New York: Basic Books.

[81] Mjolsness, E. (1994). Connectionist Grammars for High-Level Vision. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration*. Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[82] Narendra, K. S. and Annaswamy, A. M. (1989). *Stable Adaptive Systems*. Englewood Cliffs, NJ: Prentice-Hall.

[83] Newell, A. (1980). Symbol Systems. *Cognitive Science*, 4:135-183.

[84] Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

[85] Newell, A., Shaw, J.C., and Simon, H. (1963). In: *Computers and Thought*. Feigenbaum, E.A., and Feldman, J. (Ed.) New York: McGraw-Hill.

[86] Norman, D. A. (1986). Reflections on Cognition and Parallel Distributed Processing. In: *Parallel Distributed Processing*. McClelland, J., Rumelhart. D. and the PDP Research Group (Ed.). Cambridge, MA: MIT Press.

[87] Norvig, P. (1992). *Paradigms in Artificial Intelligence Programming.* Palo Alto, CA: Morgan Kaufmann.

[88] Oden, G.C. (1994). Why the Difference Between Connectionism and Anything Else is More Than You Might Think But Less Than You Might Hope. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[89] Omlin, W.C. and Giles, C.L. Extraction and Insertion of Symbolic Information in Recurrent Neural Networks. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[90] Pearl, J. (1984). *Heuristics.* New York: Addison-Wesley.

[91] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems.* Palo Alto, CA: Morgan Kaufmann.

[92] Pinkas, G. (1994). A Fault-Tolerant Connectionist Architecture for the Construction of Logic Proofs. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[93] Pollack, J. (1990). Recursive Distributed Representations. *Artificial Intelligence,* 46:77-105.

[94] Quillian, M.R. (1968). Semantic Memory. In: *Semantic Information Processing.* Minsky, M. (Ed.) Cambridge, MA: MIT Press.

[95] Rajaraman, V. (1981). *Analog Computation and Simulation.* Englewood Cliffs: Prentice Hall.

[96] Rashevsky, N. (1960). *Mathematical Biophysics.* New York: Dover.

[97] Rosenblatt, F. (1962). *Principles of Neurodynamics.* Washington, DC: Spartan.

[98] Schneider, W. (1987). Connectionism: Is it a paradigm shift for psychology? *Behavior Research Methods, Instruments, and Computers,* 19:73-83.

[99] Selfridge, O.G. and Neisser, U. (1963). Pattern Recognition by Machine. In: *Computers and Thought.* Feigenbaum, E.A. and Feldman, J. (Ed.) New York: McGraw-Hill.

[100] Servan-Schreiber, D., Cleeremans, A., and McClelland, J. (1994). Graded State Machines: Representation of Temporal Contingencies in Recurrent Neural Networks. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[101] Sharkey, N. and Jackson, S.J. (1994). Three Horns of the Representational Dilemma. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[102] Shastri, L. and Ajjanagadde, V. (1989). Connectionist System for Rule Based Reasoning with Multi-Place Predicates and Variables. Tech. Rep. MS-CIS-8906. Computer and Information Science Dept. University of Pennsylvania, Philadelphia, PA.

[103] Shavlik, J.W. and Dietterich, T.G. (1990). (Ed). *Readings in Machine Learning.* San Mateo, California: Morgan Kaufmann.

[104] Shavlik, J.W. (1994). A Framework for Combining Symbolic and Neural Learning. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[105] Shepherd, G.M. (1989). The significance of real neuron architectures for neural network simulations. In: *Computational Neuroscience.* Schwartz, E. (Ed.) Cambridge, MA: MIT Press.

[106] Shepherd, G.M. (Ed.) (1990). *Synaptic Organization of the Brain.* New York, NY: Oxford University Press.

[107] Smolensky, P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence*, 46:159-216.

[108] Smolensky, P., Legendre, G., and Miyata, Y. (1994). Integrating Connectionist and Symbolic Computation for the Theory of Language. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[109] Sowa, J.F. (1984). Conceptual Structures: Information Processing in Mind and Machine. Reading, Massachusetts: Addison-Wesley.

[110] Sun, R. (1994a). Logic and Variables in Connectionist Models: A Brief Overview. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[111] Sun, R. (1994b). A Two-level Architecture for Commonsense Reasoning. In: *Computational Architectures Integrating Symbolic and Neural Processes.* Sun, R. and Bookman, L.A. (Ed.) Boston: Kluwer.

[112] Sun, R. and Bookman, L.A. (1994). (Ed.) *Computational Architectures Integrating Symbolic and Neural Processes.* Boston: Kluwer.

[113] Turing, A.M. (1950). Computing Machinery and Intelligence. *Mind,* 59:433-460.

[114] Tanimoto, S.L. and Klinger, A. (1980). *Structured Computer Vision: Machine Perception Through Hierarchical Computation Structures.* New York: Academic Press.

[115] Tsang, E. (1993). *Foundations of Constraint Satisfaction.* New York: Academic Press.

[116] Uhr, L. and Vossler, C. (1963). A Pattern Recognition Program that Generates, Evaluates, and Adjusts its Own Operators. In: *Computers and Thought.* Feigenbaum, E. and Feldman, J. (Ed). New York: McGraw-Hill.

[117] Uhr, L. (1973). *Pattern Recognition, Learning, and Thought.* New York: Prentice-Hall.

[118] Uhr, L. (1979). Parallel-serial production systems with many working memories. In: *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.*

[119] Uhr, L. (1980). In: *Structured Computer Vision.* Tanimoto, S., and Klinger, A. (Ed.) New York: Academic Press.

[120] Uhr, L. (1984). *Algorithm-Structured Computer Arrays and Networks.* New York: Academic Press.

[121] Uhr, L. (1986). Toward a Computational Information Processing Model of Object Perception. Tech. Rep. 651. Computer Sciences Dept., University of Wisconsin, Madison, Wisconsin.

[122] Uhr, L. (1987). *Multi-computer Architectures for Artificial Intelligence.* New York: Wiley.

[123] Uhr, L. (1990). Increasing the Power of Connectionist Networks by Improving Structures, Processes, Learning. *Connection Science*, 2:179-193.

[124] Uhr, L. (1994). Digital and Analog Sub-Net Structures for Connectionist Networks. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[125] Uhr, L. and Honavar, V. (1994). Artificial Intelligence and Neural Networks: Steps Toward Principled Integration. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[126] van Gelder, T. and Port, R. (1994). Beyond Symbolic: Prolegomena to a Kama-Sutra of Compositionality. In: *Artificial Intelligence and Neural Networks: Steps Toward Principled Integration.* Honavar, V. and Uhr, L. (Ed.) San Diego, CA: Academic Press.

[127] Waterman, D.A. (1985). *A Guide to Expert Systems.* Reading, MA: Addison-Wesley.

[128] Wechsler, H. (1990). *Computational Vision.* New York: Academic Press.

[129] Winston, P.H. (1992). *Artificial Intelligence.* Boston, MA: Addison-Wesley.

[130] Yager, R.R. and Zadeh, L.A. (1994). (Ed.) *Fuzzy Sets, Neural Networks, and Soft Computing.* New York: Van Nostrand Reinhold.

[131] Zadeh, L.A. (1975). Fuzzy Logic and Approximate Reasoning. *Synthese*, 30:407-28.

[132] Zeidenberg, M. (1989). *Neural Networks in Artificial Intelligence.* New York: Wiley.

[133] Zeki, S. and Shipp, S. (1988). The Functional Logic of Cortical Connections. *Nature*, 335:311-317.

12

Connectionist Natural Language Processing: A Status Report

MICHAEL G. DYER

Computer Science Department
University of California, at Los Angeles
Los Angeles, CA 90024

1 INTRODUCTION

Connectionist networks (CNs) exhibit many useful properties. Their spreading activation processes are inherently parallel in nature and support associative retrieval of memories. The summation and thresholding of activation allows for smooth integration of multiple sources of knowledge. CNs with distributed representations (Rumelhart and McClelland, 1986) exhibit robustness in the face of noise/damage and can learn to perform complex mapping tasks just from examples. Connectionist networks are also able to dynamically reinterpret situations as new inputs are received. These features are very useful for natural language processing (NLP) and offer the hope that connectionist approaches to NLP will replace the more traditional, symbolic approaches to NLP.

Consider the task of lexical disambiguation. Words may have multiple meanings, for instance, "pot" may refer to a cooking pot, a flower pot, or marijuana. Traditional, symbolic approaches to disambiguation have utilized one of the following strategies: (a) commit to a particular interpretation and then backtrack if it is later shown to be incorrect, (b) delay the process of disambiguation until enough information is gathered so that backtracking will not occur, (c) keep track of every possible meaning, or (d) commit to an interpretation and later execute error-correction heuristics if that interpretation is shown to be wrong. None of these strategies is completely adequate. The *commit-and-backtrack* strategy is very inefficient, because backtracking will often cause much useful work to be thrown away. For instance, assume that the cooking-pot meaning of "pot" is first chosen while reading sentence S1 and then subsequent information in the sentence leads to a marijuana interpretation of "pot."

S1: The pot, which Mary bought from John, made her cough.

Here, backtracking to reinterpret "pot" (after encountering "cough") will cause the perfectly adequate analysis of the intervening relative clause to be thrown away and then redone.

The *delay* strategy has the following general problems: (a) the analysis of the rest of the sentence may not be able to go forward while commitment to a meaning is delayed and (b) no matter how long disambiguation is delayed, subsequent input may cause a reinterpretation of a given word to occur anyway — e.g., as would occur if S2 followed S1:

S2: Mary was allergic to the flower in it.

The *every-possible meaning* strategy suffers from a potential combinatorial explosion of interpretations. If there are n k-way ambiguous words in a segment of text then there will be k^n possible interpretations of that text. The *intelligent error-correction* strategy requires the specification of rules to selectively undo the harm caused by earlier commitments. As these rules execute, they will cause other inferences to be undone, which will cause yet other error-correction rules to execute, and so on. This approach requires either designing sophisticated error-correction heuristics or some kind of a general truth maintenance mechanism.

In contrast, dynamic reinterpretation is achieved as a natural side-effect of how connectionist networks operate. Activation spreads in parallel and the nodes with the most activation represent the current interpretation. As new inputs are received, the most highly active nodes may drop in activation, leading to a reinterpretation of prior inputs. Initially, the cooking-pot node will have the most activation, but "cough" will cause the marijuana node to become more active (through an activation path of smoking, etc.). Subsequent mention of being allergic will send more activation to nodes representing a different reason for coughing and, along with "flower," cause the flower-pot interpretation now to be most preferred. Thus, spreading activation exhibits aspects of all of the traditional strategies, but is implemented via a single, uniform, parallel and efficient mechanism.

The long-term goal of connectionist researchers is the complete replacement of symbolic processing models with connectionist models that exhibit efficient, robust performance and are capable of automatically learning all of the tasks that are currently programmed directly by knowledge engineers within the field of artificial intelligence. Given the many known attractive features of connectionist networks, why hasn't the connectionist paradigm already swept aside the traditional, symbolic approach? The answer lies in the fact that the symbolic processing approach retains its own attractive representational and processing features. Connectionist models, while becoming ever more powerful and sophisticated, have not yet been able to provide equivalent (let alone alternative superior) capabilities to those exhibited by symbolic systems. The rest of this paper consists of an enumeration of these symbolic capabilities, along with a description of how current connectionist networks (both localist and distributed) attempt to simulate these capabilities, and with what success (or failure).

2 DYNAMIC BINDINGS

Symbolic systems are capable of binding variables to values at run time. In symbolic systems, values may range from simple entities to complex recursive structures of arbitrary depth. Let us first consider bindings to simple (i.e., unstructured) values. For instance, the knowledge that we own what we buy might be represented by a rule something like R1:

R1: BUYS(person1, object, person2) $\longrightarrow$ OWNS(person2, object)

This rule will work for any object or person, as a result of the (typed) variables person1, object and person2 being bound to their appropriate values at execution time.

2.1 BINDINGS IN LOCALIST CNs

In localist CNs, each node represents a given syntactic or semantic entity (e.g., a predicate, such as OWNS, or a role, such as BUYER) and the amount of activation on the node represents how committed the network is to a given node (or path of nodes) as the correct interpretation of the input. However, without some kind of variable+binding mechanism, localist CNs would have to represent, before execution, all possible binding combinations, which would lead to a combinatorial explosion. For instance, for a network to conclude that Mary owns a TV (or, say, a radio) because she bought it from another person (say, Fred or Joe), then there would already have to exist CN nodes for: MARY-OWNS-TV, FRED-OWNS-TV, MARY-OWN-RADIO, etc. For just n characters and m buyable objects, there would be nxm OWNS nodes alone and an exponential number of mappings from BUYS to OWNS structures. To avoid this problem, current localist CNs use either signatures or phase synchronization to propagate simple bindings.

(1) Signatures: A signature is a unique activation value that is assigned to a given entity and that serves as a value in a binding. For instance, the CN node representing FRED might be assigned an activation value of 23 as its permanent, identifying activation (or signature) while Mary is assigned 13 and TV is assigned, say, an activation value of 7 as its signature. Signature activation is then propagated in the network along separate pathways from those used for normal spreading activation. To represent rule R1, nodes are assigned to each predicate (OWNS, BUYS) and to each role (i.e., BUYS:BUYER, BUYS:SELLER, BUYS:OBJECT, OWNS:OWNER and OWNS:OBJECT) and connections are set up between roles (e.g., from BUYS:BUYER to OWNS:OWNER) to specify how bindings should be propagated between predicates. Normal activation is propagated from BUYS to OWNS to represent a commitment to the fact that OWNS has occurred as the result of a BUYS. At the same time, signatures are propagated along role-to-role pathways. The weights on these pathways are normally set to 1 so that the values of the signatures are not altered. If the signature, say, 23 spreads from BUYS:BUYER to OWNS:OWNER and the signature 13 spreads from BUYS:OBJECT to OWNS:OBJECT, then the network can be interpreted as inferring that FRED's buying a TV resulted in FRED owning a TV. Lange and Dyer (1989) and Sun (1989, 1992, 1993) have both designed systems that make use of signature-style activation to propagate simple role-bindings.

(2) Phase synchronization: In this approach, the unit time for each basic spread-of-activation step is broken up into a few, smaller subunits of time, termed phases. For instance, if there are seven distinct phases within each spread-of-activation step, then a total of seven distinct bindings can be propagated through a localist network. In the example above, FRED, MARY and TV might be assigned phases 1, 4, and 7, respectively. The nodes in such networks are designed so that, when they receive activation within a given phase, they propagate it along their connections within the same phase of the next basic spreading-activation cycle. Phase-locking (or synchronization) was originally proposed by neuroscientists (von der Malsburg, 1981; von der Malsburg and Singer, 1987) and has been employed by vision researchers to bind distinct features (e.g., color and shape) when more than one object is in the visual field (Strong and Whitehead, 1989). Shastri and Ajjanagadde (1990, 1993) (Ajjana-

gadde and Shastri, 1989) make use of phase-locking to propagate bindings in a localist CN used to perform deductive retrieval of information.

The signature approach has the advantage that an unbounded number of signatures can be propagated simultaneously along multiple pathways while the phase-locking method is restricted to a small number of bindings. The restriction to just a few phases is not a problem for retrieval tasks, which rarely specify more than 7 unbound variables in a query. However, during natural language understanding, many new bindings can arise dynamically. For instance, within sentence S3 there are over 10 bindings that arise.

S3: The tall, thin woman bought an expensive, red sports car from the bald salesman.

Also, for n phases there are n subcycles required for each spread of activation cycle, which slows down propagation rates by a factor of n. However, both phase-locking and signature approaches suffer from the fact that only simple values can be propagated. Consider the following, common rule — one that is ubiquitous in story understanding systems:

R2: TELLS(person1, message, person2) $\longrightarrow$ KNOWS(person2, message)

The difference between R1 and R2 is that a message can be bound to any arbitrarily complex structure. For instance, if John tells Mary that a purple alien stole the hubcaps off her car, then a story understanding system should update Mary's knowledge to know this complex fact/event. In symbolic systems, this update feat is simple because all that has to be passed between TELLS and KNOWS is a pointer to the instantiated STOLE structure, say STOLE3. The STOLE3 instance will have its roles bound (i.e., that the alien is the STOLE:STEALER and the hubcaps are the STOLE:OBJECTS, etc.) as the result of parsing/analysis of the stole-related part of the text. But what would be the signature (or phase) for such a complex entity? If a STOLE3 node already exits in the network, with a unique signature preassigned, then this signature can be propagated from TELLS to KNOWS. However, in most cases the message is being mentioned for the first time by a story character; so the story understanding system must infer KNOWS from TELLS at about the same time that the stole event itself is being comprehended and incorporated into memory. Thus, any connectionist NLP system must face the problem of both dynamically creating instances of structured entities and propagating them for inferencing (e.g., so that the system can infer that Mary will be upset, etc.).

The "solution" of allowing the dynamic creation of new connectionist nodes and links at run time is not acceptable within the connectionist paradigm because it violates known constraints from neuroscience. New neurons, axons or dendrites simply cannot grow within the moments that pass during the comprehension of a sentence. Also, symbolic pointers are not acceptable because they refer to the location of a memory register and there is no evidence that any pattern of activation in one area of the brain directly encodes the location (address) of some other area in brain (as occurs with a von Neumann-style pointer.)

2.2 BINDINGS IN DISTRIBUTED CNS

Given that new nodes and connections cannot be created "on the fly" in CNs, structured bindings must be created dynamically either by modification of fast synapses (modeled as connection weights in CNs) or by changes in patterns (i.e., entire vectors) of activation over ensembles of connectionist units. Distributed CNs offer such a potential, since they manipulate patterns of activation over banks of connectionist processing units. The use of distributed patterns supports the dynamic creation of a potentially exponential number of possible values. Currently, two major methods have been developed for representing and propagating distributed patterns of activation as role bindings: tensor products and ID+Content vectors.

(1) Tensor Products : Tensor theory has been proposed for use in CNs by Smolensky (1990) and implemented in (Dolan, 1989; Dolan and Dyer, 1989; Dolan and Smolensky, 1989). Tensors result when vectors are generalized to higher ranks (i.e., a rank-one tensor is a vector; a rank-two tensor is a matrix; a rank-three tensor is a cube of units, etc.). Suppose an m-dimensional vector V represents, say, a role R, and an n-dimensional vector W represents the role's filler F. Then R and F can be bound to one another by generating the vector outer product VW, which consists of a tensor T (in this case, a matrix) of mxn elements (i.e., element-wise products of each V_i x $W_j = T_{ij}$). To extract, for instance, the binding from the role, we perform an inverse operation, such as calculating the dot (or inner) product $T \bullet W$. This approach is not as straightforward as it sounds, because a single tensor product representation will contain multiple bindings overlaid on one another. For example, to represent

BUYS(JOHN, TV, FRED) and OWNS(FRED, TV) we would set up a rank-three tensor (i.e., a cube of CN units) and overlay the following binding-triples:

[BUYS, ACTOR, JOHN]
[BUYS, OBJECT, TV]
[BUYS, FROM, FRED]
[OWNS, ACTOR, FRED]
[OWNS, OBJECT, TV]

Here, each dimension of the cube (i.e., position in a triple) represents a predicate, a role, or a role-filler. Cross-talk will result because these multiple, rank-three outer products have been overlaid (e.g., through element-wise summation) within the same tensor product. Dolan (1989) has developed various methods for extracting bindings from tensor networks in the face of such cross-talk. For example, one method is the use of "clean-up circuits." One kind of clean-up circuit consists of a network of pre-known bindings organized via inhibition. Thus, the noisy output from the tensor product is mapped to the closest matching vector (from a fixed set of possible alternatives) via a winner-take-all process. Another method is termed "pass-thru circuits." A path-thru circuit basically applies additional dot-products (representing additional constraints on what the output should look like). For example, if we are looking for the OBJECT bought and know that it is also owned, then we can set up pass-thru circuits that essentially find the intersection of two related queries, such as [BUYS1, OBJECT, ?] and [OWNS, OBJECT, ?].

Tensors are mathematically elegant and can be implemented in a CN network via conjunctive coding — i.e., multiplicative connections (Hinton et al., 1986). However, the problems of scale-up and cross-talk can be problematic. For n-dimensional vectors one needs n3 units to hold bindings as [predicate, role-name, role-filler] triples. If there are numerous BUYS events, then there will be massive cross-talk or one must encode triples that distinguish instances:

[BUYS1, ACTOR, JOHN] ...
[BUYS2, ACTOR, JOE]
[BUYS1, ISA, BUYS]
[BUYS2, ISA, BUYS].

The problem with this approach is that, ironically, the tensor network functions so much like a symbolic system that the nice features of distributed CNs (e.g., generalization) can become lost. Also, storing and access via one triple at a time creates a system that, while parallel at the subsymbolic level, is essentially sequential at the knowledge level (Sumida and Dyer, 1989; Feldman,

1989) because only one triple can be accessed or manipulated at a time. In contrast, numerous triples are activated simultaneously in a localist CN, since the predicates and roles of each event are separately represented.

(2) ID+Content Vectors: In localist CNs, each instance (e.g., BUY3) is represented by a separate node, with a connection to the general type (e.g., BUY). In distributed CNs, BUY3 will consist of a pattern (i.e., vector) of activation with segments of the pattern sharing similar activation values with the activation vector representing the type. In the ID+Content approach (Miikkulainen and Dyer, 1991), the vector is split up into two segments: (a) the Content segment, which holds information concerning t he general type of object/action being represented and (b) the ID segment, which holds information concerning the specific instance. A distributed CN, such as a PDP network (Rumelhart and McClelland, 1986), can then be trained to propagate the ID segment from one layer to another without altering the ID segment. This is accomplished by training the network on a random subset of ID patterns. Miikkulainen and Dyer (1991) have shown that PDP networks can efficiently learn to propagate novel ID patterns when trained on just a small subset of random patterns. The networks essentially learn the identity mapping for the ID segments. Miikkulainen and Dyer use this technique in the propagation of role bindings. Their system, DISPAR, contains 4 recurrent PDP networks, such as those developed by (Elman, 1990), that are connected to a lexical memory. DISPAR has the task of learning to generate complete paraphrases from fragmentary inputs of novel script-based stories (Schank and Abelson, 1977; Dyer et al., 1987. For instance, given the input fragment: "Mary ordered steak at Leone's." DISPAR generates a complete sequence of events — e. g., that includes: "Mary ate steak." Thus, generating paraphrases requires propagating bindings, since DISPAR must learn (from the training data) to perform the equivalent of inferring:

$$\text{ORDER(diner, food)} \longrightarrow \text{EATS(diner, food)}$$

A problem they encountered was that long-term knowledge (implicit in the training set of script-based stories) would be encoded in the connection weights and result in short-term information (from the input story) being overridden. For example, if every story in the training set had Mary order and eat a steak at a restaurant, then, even if (during performance) the input story contained the sentence "Mary ordered a hamburger," DISPAR would still generate "Mary ate a steak" as part of its paraphrase. This effect was also noticed in (St. John and McClelland, 1990). This problem was solved in DISPAR by representing "steak" and "hamburger" each as ID+Content vectors. The Content segment of each food-type of word contained a pattern that was similar for all foods used in the training data, while the ID segments were assigned unique patterns for each distinct food instance (e.g., hamburger vs. steak). Thus, when "Mary ordered hamburger." was input during performance, DISPAR passed the ID portion to subsequent banks without alteration. DISPAR used the pattern within the Content part (i.e., the FOOD type) to aid in processing while propagating the instance (ID segment) without change. As a result, DISPAR could conclude that "Mary ate hamburger." when told "Mary ordered hamburger." even though all training set instances consisted of Mary always ordering and eating "steak."

3 Functional Bindings and Structured Pattern Matching

Although the above techniques (e.g., signatures, ID+Content vectors, etc.) have greatly extended the symbolic capabilities of both localist and distributed CNs, they are still weak when compared to the binding capabilities of symbolic systems. The use of a heap, addressing, and pointers allows symbolic systems to create structures like STOLE3 "on the fly." Symbolic systems support propagation of even more complex bindings; for instance, they allow modules to receive entire functions or procedures as data. This results in styles of programming termed "data-driven" and "object-oriented." A simple example of this capability is the APPLY function in LISP, in which one function F1 applies whatever function F2 is passed to F1 as a parameter. This code-binding and propagation capability allows one module within a symbolic system to perform any of the operations that another module is capable of.

In addition, symbolic systems typically exhibit powerful pattern matching capabilities. A prime example is that of unification, e.g., as in Prolog. Holldobler (1990) and Stolcke (1989) have built localist CNs to perform this unification process. Holldobler, for instance, sets up several layers of threshold units. The *term layer* is a matrix of units with one side representing the terms in two expressions to be unified and the other side representing positions where terms may occur. The *unification layer* contains units that are connected in a manner to impose unification constraints, for instance, whether two occurrences share a common variable. The *occur check* layer makes sure that cycles do not occur, such as x becoming bound to f(x).

The major problems with this localist CN approach to unification are: (a) that all of the units and their connections must be prewired for the given expressions that are to be unified and thus are finite and non-general, and (b) only a single solution is produced. In contrast, unification in logic programming languages, such as Prolog, work on an infinite number of possible expressions to be unified and can return multiple solutions when they exist. Thus, a localist CN with any generality would require a method for recruiting and wiring up units dynamically. In the area of distributed CNs there are no architectures, to my knowledge, designed to attempt unification.

4 ENCODING AND ACCESSING RECURSIVE STRUCTURES

Recursive structure is essential for high-level reasoning, particularly natural language processing. Localist networks can represent recursive structures by connecting up the appropriate nodes in a tree-like manner. However, localist networks that propagate phases or signatures have difficulty with propagating multiple instances of the same type. Consider sentence S4:

S4: John told Mary that Betty told Fred that Jim went home.

Here there are two TELL structures. In localist CNs, there is usually only one node for each type of predicate. Thus, the "John told Mary" segment can be represented by passing signatures (or phases) over the TELL:TELLER and TELL:RECEIVER nodes. However, the embedded "Betty told Fred" must be represented by another TELL instance that would be dynamically bound to the TELL:MESSAGE node of the top-level TELL. One solution to the problem (of multiple instances of the same type) is to have n copies for each predicate (and corresponding roles). If n = 2, then the CN network could parse and represent one TELL instance embedded within another TELL. For instance, if each TELL node had a pre-assigned signature, then the signature of the embedded TELL could by propagated to the TELL:MESSAGE of the outer TELL. This approach will always fail for sentences with embeddings greater than n. This limit, on depth of recursion, may not be so bad, however, because people also exhibit a limit. Consider S5:

S5: John told Mary that Betty told Fred that Sally told Frank that Jim went home.

Most people, upon hearing such a sentence out loud, protest that they cannot keep straight who is telling whom and immediately recall it as "Several people telling other people that Jim went home.."

In the area of distributed CNs, early PDP networks lacked a recurrent layer; as a result, the encoding of recursive structure was problematic. If a 3-layer PDP network, for instance, had 4 banks (e.g., representing the ACT, ACTOR, RECIPIENT, OBJECT) then a sentence like S4 could not be encoded, because the embedded TELL required its own ACT, ACTOR, etc. This problem has been solved by the use of distributed CNs with a recurrent layer. A number of distinct recurrent architectures have been employed to encode recursive structures and their constituents. Two common approaches are the Simple Recurrent Network (SRN) of Elman (1990) and the Recursive Autoassociative Memory (RAAM) of Pollack (1988, 1989, 1990). In SRNs, the hidden layer is copied onto an added bank in the input layer (termed the "context" bank) and then is fed back into the hidden layer at the next cycle. In contrast, RAAMs make use of an autoassociative (or encoder) network, in which a PDP network is trained to generate on the output layer the same pattern as that placed on the input layer (Rumelhart and McClelland 1986). In Pollack's RAAMs, the pattern of activation produced on the hidden layer is copied back into a bank on both the input and output layers. Figure 1 illustrates how S4 can be encoded in a RAAM.

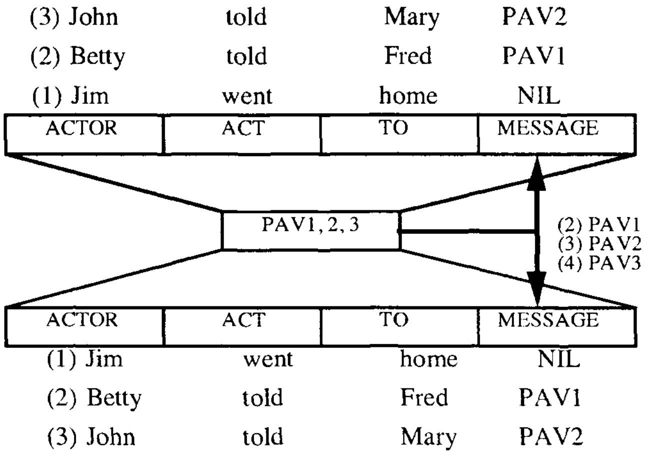


Figure 1 Encoding of an embedded structure within a RAAM.

First, "Jim went home" is autoassociated on the RAAM's input/output layers. The resulting pattern of activation (PAV1) in the hidden layer is then placed in the input/output banks representing the MESSAGE role. Then "Betty told Fred PAV1" is autoassociated. The resulting pattern over the hidden layer (PAV2) is placed in the MESSAGE bank. Now "John told Mary PAV2" is autoassociated. The resulting hidden-layer activation vector (PAV3) now encodes the entire recursive structure. To retrieve this recursive structure, one can place PAV3 on the hidden layer of the same RAAM and [JOHN TOLD MARY PAV2] will be reconstructed on the 4 banks of the output layer. Now PAV2 can then be placed over the hidden layer and [BETTY TOLD FRED PAV1] will be reconstructed on the output layer. When PAV1 is placed over the hidden layer, the pattern for [JIM WENT HOME NIL] will appear in the banks on the output layer. So a RAAM can basically function as a stack thus can encode both simple lists and trees into a fixed-width vector. For example, Miikkulainen (in press) makes use of a RAAM to act as a stack in a distributed CN that learns to parse embedded relative clauses.

If a long-term memory is added, to store these hidden-layer patterns (e.g., PAV1, PAV2), then autoassociative networks can be used to store graphs (i.e., recursive structures with cycles). For instance, Dyer et al. (1992) made use of an architecture called DUAL, which consists of a 3-layer PDP network (labelled STM) and an autoassociative encoder network (labelled LTM) whose hidden layer is of the same length as the input/output layers of the STM and whose input/output layers are of a length equal to the number of weights in all STM layers. DUAL has been used to encode a simple semantic network (i.e., a graph of nodes and labelled links, with cycles). For instance, each node is defined as a number of labelled-arc-to-node pairs:

JOHN:	-LOVES⟶	MARY
	-GENDER⟶	MALE
	-JOB⟶	PROFESSOR
	...	...

The STM's weights are set (via backpropagation learning) to associate roles with values (e.g., LOVES on input layer and MARY on the output layer). After learning, all of the *weights* in the STM network are then passed as a single, (larger dimensional) vector of activation values (call it V-JOHN) to the input and output layers of LTM, which is taught to autoassociate it. Thus, LTM stores entire STM networks. To retrieve the V-JOHN STM weights, one places V-JOHN on the LTM's hidden layer. The resulting weights (on LTM's output layer) can then be used to reset the weights of the STM. To retrieve any piece of information about JOHN, we now place the appropriate role representation (e.g., JOB) on the STM input layer and its value (e.g., PROFESSOR) will appear on the output layer. Now, consider the encoding of cycles. Suppose that MARY has the following arcs:

MARY:	-LOVES⟶	JOHN
	-GENDER⟶	FEMALE
	-JOB⟶	DEAN

Here we have a cycle because both [JOHN -LOVES⟶ MARY] and [MARY -LOVES⟶ JOHN]. To train STM to encode MARY, we train STM to associate MARY's roles with the appropriate values. For instance, we place LOVES on the STM input layer and V-JOHN on the STM output layer. After training, the resulting STM weights (call it V-MARY) now encode all information about MARY. However, when the JOHN network was encoded, we did not have V-MARY as the representation for MARY (we had just whatever initial, arbitrary representation had been selected to represent MARY). So now we have to retrain the JOHN STM network to properly associate LOVES with V-MARY. This encoding cycle will alter the STM weights (that encode all properties for JOHN), resulting in a new set of weights (call this vector of weights V-JOHN1). As a result, the encoding for MARY must be altered (since MARY now -LOVES⟶ V-JOHN1, not V-JOHN). MARY is also now better represented by a new vector (call it V-MARY1), and so on. Over time, the network will find an encoding of both JOHN and MARY as their distributed representations are recirculated through the DUAL architecture. If two nodes N1 and N2 have similar arc/node associations, then N1 and N2 will end up being represented by vectors that are very similar. This similarity aids in generalization. For instance, if a country C1 has n properties (where, say, one property is [C1: -PRODUCES⟶ RICE] and country C2 has n-1 properties that are the same as those of C1 (but it is not known whether or not C2 produces rice) then the similarity of the vectors formed for C1 and C2 will cause DUAL to conclude that C2 also produces rice.

5 Forming Lexical Memories

Natural language processing requires a lexical memory. In symbolic systems, each word is encoded as a symbol (e.g., in ASCII) that is mapped to some frame-like structure (Minsky, 1985) with attached rules. For example, in the BORIS story understanding and question-answering system (Dyer, 1983), the word "eats" is mapped to an INGEST frame with a number of rules (implemented as test/action "demons"). For instance, one of the demons searches for a FOOD frame following the INGEST frame and if found, the demon binds the FOOD frame to the OBJECT role of the INGEST frame. Another demon searches for an animate agent preceding the INGESTS frame and binds it to the ACTOR role, and so on.

In such a system, the internal (ASCII) representation for the word "eats" is arbitrary and static. In localist CNs, the node for "eats" is also static and its connections (to other nodes representing words or frames) are specified by the knowledge engineer. In contrast, in some recent distributed CN architectures, methods have been developed to automatically form distributed representations (i.e., activation vectors) for lexical entries. The most interesting and useful result of these methods is that words with similar semantics end up possessing very similar representations (e.g., as result in the recirculation method in DUAL) — thus supporting generalization to novel yet related natural language texts. For instance, if "pasta" ends up forming a similar representation to "spaghetti" then a distributed connectionist network trained on "John ate the spaghetti" will automatically tend to correctly process "John ate the pasta" even if it has never been trained on this particular input.

Two methods have been developed for automatically forming lexical representations: Miikkulainen's FGREP method (Miikkulainen and Dyer, 1991; Miikkulainen, 1993) and Lee's xRAAM method (Lee, 1991; Lee et al., 1990; Lee and Dyer in press). In the FGREP method, one PDP network (call it M1) is trained to map words (represented as activation vectors) or word sequences (if the network is recurrent) from banks in the input layer to banks in output layer. For instance, "chicken" might map from the SUBJECT input bank to the ACTOR output bank in "The chicken ate the worm." while it might map from the DIECT-OBJECT input bank to the RECIPIENT output bank in "The man ate the chicken." While the weights in the M network are updated (via backpropagation) to learn the correct mapping, at the same time, the vector (representing "chicken") is modified. This modification is accomplished by extending backpropagation learning over a set of weights representing "chicken" in a lexical memory. All other weights (representing other words) in the lexicon are not modified. As a result, the representation of any word W will become altered as the network M is trained to map a word W from its input to output layers. A single lexicon can then be linked to multiple PDP networks. As each network learns to map words from the lexicon, those words will be altered and their altered weight vectors are stored back into the lexicon. Thus, as each network as trained on lexical data, the representations of the training data are themselves undergoing alteration. Figure 2 illustrates the FGREP process on a recurrent network.

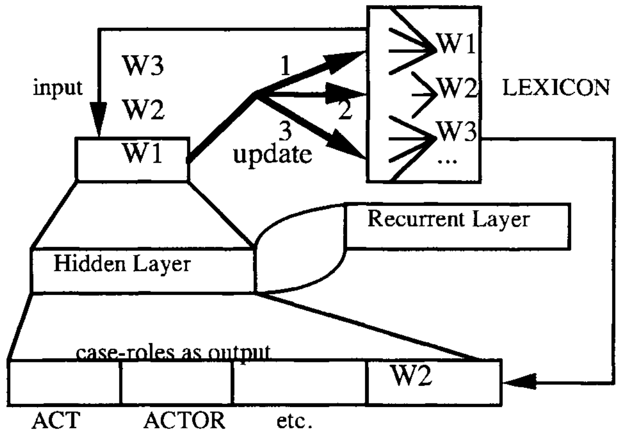


Figure 2 FGREP process. Word representations are taken from the lexicon and used to train the input/output layers of a recurrent network. During learning, backpropagation is extended back into just that section of the lexicon that represents the current input word and just its weights are updated at that point. The altered representations in the lexicon can also be used to train other networks, which will cause these words to again undergo modification.

Interestingly, convergence to stable patterns does not take much longer than training with static data. The reason is that the alterations in the representations of the lexical data make the mapping tasks easier — i.e., for the networks that are learning to map this data from their input to output layers. That is, the network M has an easier learning task because the data being used is being altered to support M's mapping task. Using FGREP and ID+Content vectors, Miikkulainen and Dyer (1991) designed DISPAR, a story paraphrasing system consisting of 4 SRN modules and a lexical memory. Each module performs a distinct task: (a) mapping a sequence of words to a case-role event representation, (b) mapping a sequence of events to a script representation, (c) mapping a script back to an event sequence, and (d) mapping an event to a sequence of words. During training, each module is trained with word representations taken from the lexicon and modified via the FGREP method. During performance, DISPAR is given, as input, a script-based story fragment and generates, as output, a complete story — i.e., with all intervening actions and roles instantiated.

In the xRAAM method (Lee, 1991; Lee et al., 1990; Dyer and Lee, in press), a distributed representation of each word is formed by encoding all propositional information in which the word is involved. An xRAAM network is a RAAM augmented with a lexical memory. Consider the word "milk." This word might be represented in terms of the following (simplified) propositions:

[MILK IS WHITE]
[MILK PRODUCED-BY COWS]
[MILK CONTAINED-IN CARTONS] ...

As each proposition is autoassociated within a RAAM (as described earlier), a pattern of activation is formed on the hidden layer. The final pattern formed is taken to be the representation of the word/symbol being encoded (in this case, MILK). By cycling back through the RAAM, this propositional information can be extracted (as described earlier for autoassociative networks). This distributed representation (i.e., as an activation vector) is stored in a separate lexicon. The representations for other words are formed in the same way. For example, COW will be involved in the following propositions:

[COW PRODUCES MILK]
[COW EATS GRASS]
[COW HAS FOUR-LEGS] ...

After COW is encoded in a RAAM and its lexical representation has been formed, we must then go back and re-encode MILK (because when [MILK PRODUCED-BY COWS] was encoded into the RAAM, the representation for COWS was different). Thus, the encoding process involves a recirculation of all words (as in the DUAL and FGREP methods) in which the fact that there are changing lexical representations cause other modules to have to be retrained on these new lexical representations (Dyer, 1990). Lee terms the resulting representations Distributed Semantic Representations (DSRs) because: (a) they are distributed patterns that can be passed to a variety of CNs, (b) they encode the propositional content of the words and this content can be extracted by different modules (who were not necessarily involved in the learning of the word's representation), and (c) this "symbol recirculation" process (Dyer, 1990) results in DSRs with similar meaning having similar vectors (as with the FGREP and DUAL methods). Using DSRs, Lee designed the DYNASTY system (Lee, 1991; Dyer and Lee in press) — a multi-modular system of PDP networks, SRNs, and xRAAMs — which takes simple goal/plan-based stories as input and generates, as output, a chain of inferred goals, plans and/or sub-goal preconditions as an explanation for actions taken by the main narrative character.

6 FORMING SEMANTIC AND EPISODIC MEMORIES

Episodic memory (Tulving, 1972) consists of personal episodes or events and is distinct from semantic memory, which consists of general world knowledge. In symbolic systems, both semantic and episodic memories are built out of symbols. The knowledge engineer selects the basic set of symbols to use. Episodic memory then consists of instantiations of semantic memory symbols. For instance, semantic memory in the BORIS system (Dyer 1983) contained an INGEST structure in semantic memory. Specific INGEST instances were indexed in episodic memory as the result of reading a story involving a particular character eating a particular food at some particular location or time.

In localist CNs, semantic memory consists of a connected network of nodes, also specified by the knowledge engineer. The formation of episodic memories is problematic in localist CNs because any dynamically-created instance is represented by semantic nodes momentarily containing signature (or phase-locked) activation. This activation must be cleared from the network before the next sentence is read. But then how are any event instances to be stored away for long-term retrieval? One way is to simply created new nodes and links, but as we have seen, this approach violates a connectionist paradigm constraint. Thus, any localist CN theory of episodic memory will require a theory of how to "recruit" preexisting nodes and connections to form new, long-term memories.

In distributed CNs, the formation of semantic memories is straightforward. Semantic memory is represented in the weights of the network. These weights undergo modified (e.g., via backpropagation) to reflect statistical features inherent in the training data. However, the storage and retrieval of specific episodes is problematic because events are laid on top of one another in a distributed connectionist network (since the same network performs multiple mappings). This approach supports generalization but makes the retrieval of individual events difficult. To date, the most successful approach to modeling episodic memory has been to make use of extensions of Kohonen self-organizing feature maps (Kohonen, 1988). A feature map consist of a 2-dimensional plane of units, with each unit receiving, in parallel, the same n-dimension vector as input along its weights. The most active unit on the map then causes its neighboring units to modify their weight vectors so that, in the future, they will respond more to that input. The result of this form of learning is that similar inputs will tend to activate contiguous regions. Thus, a Kohonen feature map clusters or self-organizes the input data in a 2-D space without the need for explicit training (as in backpropagation). Feature maps have several nice properties, including: (a) Similar events will be stored in similar regions, thus supporting generalization. (b) Distinct vectors will be mapped to different regions and thus be retrieved without interference from other memories. In general, humans recall very distinct actions/objects more easily also. (c) Recent memories get laid on top of other and thus are more memorable. Humans also demonstrate this kind of recency effect.

The DISCERN system (Miikkulainen, 1993) is an extension to DISPAR that includes a question-answering module that learns to retrieve unique events from an episodic memory. The episodic memory consists of Kohonen features maps that are organized into a 3-level hierarchy. At the top level, different scripts are self-organized on the map (e.g., restaurant vs. travel vs. shopping). At the middle level are maps that self-organize distinct tracks within a given scripts (e.g., for restaurants, the tracks are fastfood vs. cafeteria vs. snazzy restaurant, etc.). At the bottom level are the unique bindings (e.g., within the fastfood track of the restaurant script, the diner was Joe and the food was steak). Hierarchical maps were employed to speed up learning because, in Miikkulainen's task domain, the data itself is hierarchical. Another extension Miikkulainen made was to alter feature maps so that they could store bindings. Standard Kohonen maps simply categorize their data. Miikkulainen altered the bottom level feature maps so that role bindings are encoded in the lateral connections (i.e., between nearby units on the map).

How does this approach compare to that use in symbolic systems? Kolodner (1984) developed a symbolic, computational model of human episodic memory. Her system, CYRUS, modeled aspects of the episodic memory of Cyrus Vance (when he was Secretary of State). CYRUS contained episodes described in the press, e.g., trips to foreign countries, summit meetings, treaty negotiations, etc. Unlike hierarchical Kohonen maps, which have a fixed-size per map and hierarchical depth, Kolodner's memories consisted of multi-indexed symbolic structures of arbitrary depth. However, Kohonen maps have the ability to encode finer regions within areas of the same map, and so can encode hierarchical structure.

Kolodner also modeled a complex set of heuristics for generating retrieval cues automatically — i.e., to search memory in those cases in which indices did not exist directly. For example, CYRUS did not have an index of wives-meeting-wives, yet it could still recall times that Vance's wife met Menachim Begin's wife, by generating possible retrieval cues (e.g., trips to Israel in which wives are taken along and embassy parties). No such meta-level knowledge (i.e., of how memory is indexed) yet exists or is employed in connectionist models. However, this lack is probably more due to the youth of the connectionist NLP field, which only began to develop in the late 1980s. To my knowledge, Miikkulainen's model is the first to even attempt to model episodic memory. The advantage of the connectionist approach is that the resulting memory exhibits well-known connectionist features, namely, it is robust to noise/damage and provides parallel, associative retrieval from subsymbolic cues.

7 ROLE OF WORKING MEMORY

In addition to lexical, semantic and episodic memories, there is a need for a limited capacity but rapid memory — in which intermediate structures can be built and manipulated. For example, in Touretzky and Hinton's (1988) distributed connectionist production system (DCPS), a coarse-coded (Rumelhart and McClelland, 1986) working memory is used to store sets of triples. Production rules are then "matched" (via spreading activation) to determine which rule to fire next. In Barnden's connectionist implementation of Johnson-Laird's model of syllogistic reasoning (Barnden 1991; in press; Barnden and Srinivas, 1991; Johnson-Laird, 1983) a central component is a connectionist working memory (termed Conposit) in which instances of predicates (representing objects, events, etc.) can be rapidly represented and bound to one another. The method, by which this rapid binding is performed, is unique. Barnden employs a unique approach to representing transient bindings, termed *relative-position encoding*. Conposit's working memory consists of a 2-dimensional matrix where each cell in the matrix consists of a complex subnetwork capable of a number of operations. One of these operations is to notice configurations of activation patterns in neighboring cells. Such operations allow symbols (represented as activation patterns within a given cell) to be bound simply by being placed in any one of the 8 contiguous cells surrounding a given cell. This 8-cell region places an upper limit on the number of symbols that can be bound into a single structure, so Barnden employs an additional binding mechanism, termed Pattern Similarity Association. Namely, structures are bound to one another if they share the same symbols. Each cell of the matrix is quite complex and can perform numerous operations. Barnden argues that such complexity is needed to model human syllogistic reasoning.

Another situation requiring working memory is in the dynamic linking of recursive structures. For example, in efficiently parsing embedded relative clauses, Miikkulainen (this volume) employs a RAAM to act as stack, in order to push/pop clauses appropriately as they are parsed into case-role vectors.

8 ROUTING AND CONTROL

Both localist and multi-module distributed CNs require methods for controlling the sequencing of operations and for routing information among different modules. Localist networks must control along what pathways signatures are to travel. For example, in the ROBIN system (Lange and Dyer, 1989; Lange, 1994), there are nodes whose connections act to control or gate connections between other nodes. If ROBIN receives "John inhaled the pot" as input then activation will not spread at all to the FLOWER-POT meaning of "pot," because gating nodes will only let through signatures within a given set (i.e., in this case, different types of gases). Thus, gating acts to greatly reduce the amount of spreading activation that occurs and to impose syntactic and semantic restrictions on inference propagation.

In most distributed CNs, control processes are specified procedurally. For example, in DISCERN all routing of patterns between modules, and control of when modules execute (during both learning and performance), are specified by designer-specified procedures.

Miikkulainen (1994), however, has shown how control can be learned automatically. His SPEC model, designed to parse embedded relative clauses, contains 3 modules: (a) a SRN which takes as input an sentence with embedded clauses, (b) a RAAM (which acts a stack) and (c) a three-layer PDP network, termed the Segmenter. It is the job of the Segmenter to determine when to push or pop the stack, based on the current state of the parse (i.e., due to encountering clause boundaries for right branching vs. left branching vs center embedded clauses). Miikkulainen's work shows that distributed connectionist architectures can be trained to control their operations instead of having to employ a top-level, non-connectionist procedure.

With respect to all CNs, one can imagine a "granularity spectrum." At one end of the spectrum are purely localist CNs, with potentially many thousands/millions of nodes, each representing an individual type or instance. Here each single node acts as a module; the grain size is very small and the number of modules is extremely large. At the other extreme of the granularity spectrum lies, say, a single SRN, where each layer consists of an extremely wide vector. Here the granularity is very coarse, with only 1 module (consisting of 3 layers with one set of recurrent connections). The problems with the localist extreme are: (a) a combinatorial explosion of nodes are needed to represent world knowledge — e.g., the problem of representing all possible visual angles and other information concerning one's grandmother via separate "grandmother neurons" (Feldman, 1989), (b) the difficulty of incorporating learning, and (c) the recruitment of neurons/connections in forming long-term memories dynamically. The problems with the other extreme (i.e., single SRN) are: (a) learning to set the weights (especially when performing complex task involving language and higher-level reasoning) will take an impossibly long time, and (b) all brains exhibit a lot of specialization of circuitry and modularization (even if the modules are heavily overlapping). Miikkulainen and Dyer (1990) have shown that breaking up the the story paraphrase task into 4 modules (each independently trained yet communicating via a common lexicon) dramatically reduces the overall training time. This approach is an obvious — i.e., one of divide and conquer. But the DISPAR and DISCERN modules still lie very much near the single module extreme of the spectrum, since they contain under a dozen, relative large modules. Can we imagine architectures with modules that are finer than those in DISCERN, perhaps hundreds or a few thousand, but far fewer than those at the localist extreme? This level of granularity would correspond more to how the brain appears to be organized.

The DCAIN system (Sumida, 1991; Sumida and Dyer, 1989, 1992) lies within this region of the spectrum. It is a distributed CN which consists of (potentially many hundreds of) ensembles of units. The global organization between ensembles is like that of a semantic network. Thus, these networks are termed Parallel Distributed Semantic (PDS) networks. Each ensemble is connected to other ensembles via multiple adaptive connections which are themselves under the control of learnable routing ensembles, termed propagation filters (Figure 3).

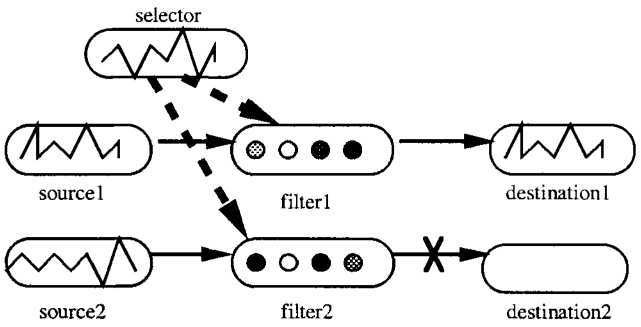


Figure 3 Propagation filter arrangement. A pattern (jagged lines) over the selector ensemble causes one filter ensemble to go above threshold and allows routing of a pattern from source1 to destination1 while blocking the propagation of other patterns. Arrows represent full connectivity.

A propagation filter consists of a selector ensemble of units and a filter ensemble. When a particular pattern of activation occurs over the selector ensemble, every unit in the filter group is driven above threshold, which allows an activation vector to be routed (over multiple connections) from a source ensemble to a given destination ensemble.

In DCAIN, each distinct type of semantic or syntactic information is represented as an ensemble of connectionist units. Each ensemble is connected to other ensembles based on semantic/syntactic relations between types. For example, the predicates BUYS and OWN and their roles would be represented each as a distinct ensemble. To represent the implication that [x BUYS y $\longrightarrow$ x OWNS y], the roles of the BUYS ensemble are connected to the appropriate roles of the OWNS ensemble. Thus, at the ensemble level, PDS networks share organizational principles with localist networks. However, there are no separate instance nodes. Instead, each instance of a type (e.g., BUYS1 — say, that John bought a TV from Sears) is represented as a particular pattern of activation that occurs within the BUYS ensemble. Thus, an exponential number of instances can be stored and the problem of node recruitment (at least for representing instances) does not arise. The relationship between a predicate ensemble and its role ensembles is similar to that of an autoassociative encoder network. That is, there are connections from role ensembles to the predicate ensemble and back and these connections are dynamically modified (during learning) so that the activation pattern over the predicate ensemble will cause the reconstruction of the correct role ensembles and vice versa. For example, if the pattern for BUYS1 is placed in the BUYS ensemble, then that pattern will cause the role ensembles to reconstruct (via pattern completion) the following values: BUYER = John and FROM = Sears. Thus, unlike localist CNs (which have relationships between role values and predicate instances specified by hand), PDS networks learn this relationship. Also, PDS networks can store more than one type within a given ensemble. For instance, BUYS could be a pattern of activation over an ensemble designated to be an ACT ensemble, while OWNS might be a pattern of activation over a more general STATE ensemble. Thus, the ACT ensemble might hold other actions (besides buying) and the STATE ensemble might hold other states (besides the OWN state). Syntactic categories (e.g., SUBJECT, DIRECT-OBJECT) are also represented as ensembles and a parsing analysis that, say, John is the subject of a sentence, is represented by propagating the John pattern of activation to the SUBJECT ensemble. Relationships among syntactic and semantic pieces of knowledge are represented in terms of propagation filters, which determine how patterns are propagated among ensembles. Figure 4 illustrates syntactic/semantic analysis of simple sentence, i.e., of the form [SUBJECT, VERB, D-OBJECT]).

Notice that, in Figure 4, the pattern for "boy" controls its own routing, i.e., causing it to be routed to the humans ensemble (vs. the animals ensemble). The pattern in the humans ensemble can only be routed to the subj ensemble after the correct kind of noun-phrase pattern has arrived in the NP ensemble. The correct pattern in the NP ensemble will cause patterns for "the" and "boy" to be reconstructed in the DET and N ensembles. The correct pattern in the Basic-S ensemble will cause its appropriate roles (in this case, subj = boy, verb = hit, dir-obj = dog) to be reconstructed in its role ensembles. When "the cat" is input, existing patterns over the NP and verb ensembles will cause "cat" to be routed to dir-obj (vs. to subj). This part of the analysis is not shown in the figure. Note that each predicate/roles encoder network and selector/filter group is trained to perform its reconstruction (or routing) task.

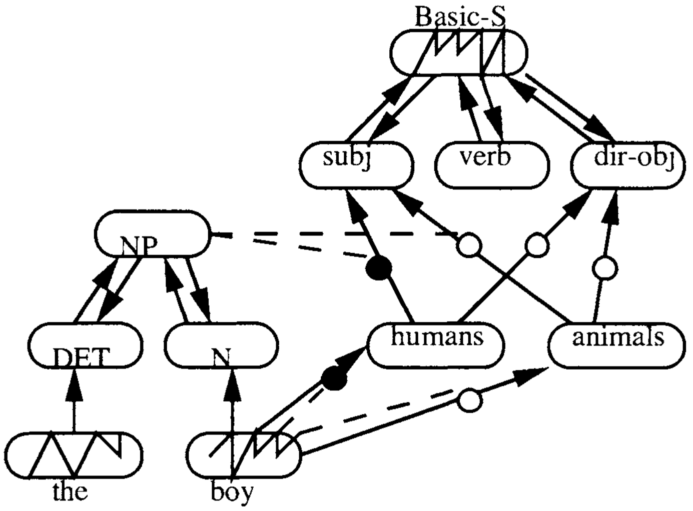


Figure 4 A fragment of a simplified PDS network showing some of the interaction between syntactic and semantic elements when parsing the phrase "the boy" from the sentence "The boy hit the cat." Propagation filters are small circles with black color indicating filters that allow allow patterns to be propagated. Two-way arrows between predicate and role ensembles (ovals) represent autoassociative encoder networks, with a predicate serving as a hidden layer and the roles serving as both input and output layers (i.e., the output layer is "folded" back onto the input layer). Dotted lines are from ensembles (that act as selectors) to filters.

The word "hit" has different interpretations, depending on context. For example, it can mean to perform music, as in "The boy hit the note." Figure 5 shows how filters propagate role bindings, based on the context within which "hit" appears.

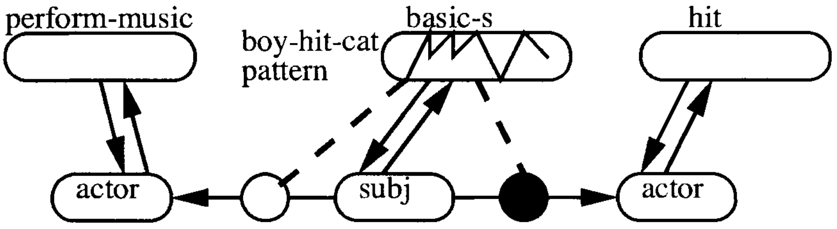


Figure 5 Pattern-based routing to perform word disambiguation. The pattern for boy-hit-cat appearing over the basic-s ensemble causes a filter to open and propagate the subject of the sentence to the actor role of the hit ensemble.

PDS networks retain many of the advantages of both localist and distributed CNs. Training PDS networks is more rapid than training distributed CN architectures with just one (or a few) modules because each PDS subnetwork is small and can be trained independently. In general, more modules of smaller size will result in faster overall training when implemented over a parallel architecture. Since there are many modules in separate locations, it is possible to pursue in parallel many distinct inference paths at the knowledge level. Novel instances can be created dynamically by forming new patterns over existing ensembles. Since role ensembles are connected to predicate (type) ensembles in the manner of encoder networks, they have the ability to perform pattern completion (i.e., roles reconstructing its predicate instance or a predicate instance reconstructing its roles) and to generalize to related patterns. As a result of pattern completion, PDS networks can propagate structured bindings — i.e., a pattern laid over a type ensemble will cause the reconstruction of all of its role bindings in the associated role ensembles. These role patterns can then be routed to other predicate ensembles, thus causing their roles to be reconstructed, and so. If there is only one ensemble for a given predicate then only one instance (e.g., only TELL1 or TELL2, but not both) can be active at a time. However, it is possible to sequence through these instances over time (e.g., first filling the N ensemble with "boy" and then later with "cat" in "The boy hit the cat"). This sequencing is controlled by propagation filters. Finally, because predicate/role ensemble groups are trained via example to act as encoder networks, they extract statistical regularities from the training data. Their distributed representations also allow them to exhibit robustness in the face of noise and/or damage because the loss of a unit within an ensemble will only degrade its performance, not destroy it. In addition, PDS networks reside in a region of the "granularity spectrum" that is closer to that of the brain (i.e., than either purely localist or distributed CNs with just a few modules).

9 GROUNDING LANGUAGE IN PERCEPTION

Although language relies on (and manipulates) highly abstract concepts and other forms of knowledge, it appears to be the case that children acquire early language semantics by associating verbal utterances (e.g., from adult caregivers) with ongoing sensory/motor experiences. Consider (one meaning of) the word "passing." After the child has learned simple objects by verbal/visual association (e.g., "ball," "car," "dog") the child can begin to learn the meaning of "passes" (e.g., as in "the car passes the dog") by simultaneously observing a car moving along, coming up from behind the running dog and then outstripping the dog, with both moving in the same direction. At the same time the child hears the phrase "car passes dog" (or, in the case of deaf children, receives a gestural sequence as visual input). By watching different size/shape/color objects catch up to and pass one another, the child can begin to form a perceptually based representation of the word "passes." It is unclear what a candidate symbolic representation would be. More likely, a major part of the meaning of "passes" consists of a generalized spatio-temporal visual experience. This perceptually based representation can then serve as a foundation for more abstract representations (such as "passes" later meaning that one becomes superior to someone else in a given cognitive skill, like playing chess, or that one "passes" an exam, etc.).

The task of mapping the abstract symbols of language to/from perceptual/motor experience has been called variously, the "symbol grounding task" (Harnad, 1990), "L_0 language acquisition task" (Feldman et al., 1990) or "perceptually grounded language learning task" (Nenov, 1991; Dyer and Nenov, 1993). This task has been addressed by several connectionist researchers.

Regier (1992) developed a connectionist network that learns the meanings of phrases by associating them with two simple objects (where one is a stationary landmark and the other is moving relative to it in a 2-D microworld). This research is part of the L_0 Project led by Feldman (Feldman et al., 1990) at the International Computer Science Institute at Berkeley, CA. The long-term goal of the L_0 Project is to acquire language via association with perception. The architecture is designed to extract object features (e.g., center of mass and major axis orientation) and spatial features (e.g., concerning the relative angle, orientation, and distance of the moving object with respect to the landmark). These spatial features are extracted by non-neural, procedural modules. The resulting representations consist of both feature vectors and 2D feature maps. The feature maps are trained to produce, on the output layer, descriptions of the motion sequences. The learning method is a variant of backpropagation learning, in which every positive instance (during training) for a given spatial concept constitutes weak, implicit negative evidence for all other spatial relationships being learned. Regier argues that this modification allows the system to learn from positive examples only. He points out that in the child acquisition data, children acquire language apparently without the benefit of negative evidence (Pinker, 1989). Regier's system consists of distinct and independent modules, each with a different connectivity arrangement and learning/activation parameters.

The DETE system (Dyer and Nenov, 1993; Nenov and Dyer in press-a, b, c) also learns the meanings of word sequences via association with simple moving objects. The DETE system's microworld (called Blobs World) consists of a simulated (64x64 "pixels") visual screen (VS) of up to five 2-D, homogeneous, mono-colored (and possibly noisy) "blobs" of various shapes (e.g., rectangular, circular, triangular). During learning, DETE receives also a simulated verbal sequence describing the visual sequence. Motor sequences may also be input, which tell DETE how to move and/or zoom in/out its single EYE. DETE also has a FINGER which can be made to touch or push blobs. After learning, DETE performs two tasks: *(a) Verbal-to-visual/motor association* — given a verbal sequence, DETE generates the visual/motor sequence being described. (b) *Visual/motor-to-verbal association* — given a visual/motor sequence, DETE generates a verbal description of it.

The current version of DETE is a massively parallel model that consists of over 1 million virtual processors, executing on a 16K processor CM-2 Connection Machine. Interface modules (i.e., that map simulated visual/verbal input to learning/memory subsystems) are parallel, array-processing (non-neural) procedures, while internal processing/memory modules themselves are modeled as highly structured neural networks modules (termed katamic memory) with each composed of novel neural elements.

Like a child, DETE must be taught incrementally. In a series of learning experiments DETE was first taught the names of blobs by being given scenes of blobs with a single shape, but with varying colors, sizes, locations and motions. As a result, DETE extracts what is invariant (i.e., shape) and forms the strongest associations between verbal input (e.g., "circle") and its internal representations for size, shape, etc. DETE next learned the meanings of words for color, size and location with respect to center of the VS (e.g., "above," "right," "in-center," "far," etc.). DETE then learned single words for actions/events. Such words include: "moves," "accelerates," "turns," "bounces," and "shrinks" (i.e., change in blob size). Once these words were learned, DETE was tested by presenting it with verbal input only, and DETE indicates its comprehension by generating internal representations of this visual behavior. DETE's syntactic ability is currently limited to extracting word preference order (e.g., that size terms come before color terms) and the most complex sentences it has learned are of the sort: "big red ball moves diagonally down ... bounces ... moves diagonally up."

In DETE all visual/motor input is mapped (by non-neural interface routines) to *regions of active neurons* over a set of *Feature Planes* (FPs). The 5 *visual* FPs are: Shape (SFP), Size (ZFP), Color (CFP), Location (LFP) and Motion (MFP). Each FP is composed of a 2-D array of 16 x 16 (256) neurons. Different active regions within a Feature Plane represent different values for that feature. An *active* neuron is one that oscillates, i.e., it fires periodically (with output 1) and is silent the rest of the time (with output 0). FPs have either a raster-linear or *topographic* layout. For instance, the LFP and MFP have topographic layouts. If a blob is in the lower right corner of the VS, then its position will be represented by a region of active neurons in the lower right corner of the LFP. On the MFP, the speed of a blob is represented by distance from the center, with stationary objects at the center and more rapidly moving objects toward the periphery. There are also FPs for FINGER and EYE dynamics. Figure 6 shows a (simplified) sequence of images on the VS, along with the visual representations that are produced (by array processing procedures) over a subset of the Feature Planes.

DETE makes use of phase locking to handle the "feature binding problem." For example, if both a big-region and small-region of the size FP are active and also both a square-region and circular-region of the shape FP, then how is DETE to distinguish whether what is being represented is: (a) a small circle and a large square or (b) a small square and large circle? DETE solves this feature-binding problem by breaking down its basic processing cycle into phases and assigning a distinct phase to each blob. Thus, if it is a small circle and large square that is on the VS, the active small-region and active circular-region will both be firing with the same phase. This phase difference is represented pictorially in Figure 6 as distinct textures (with active regions for the same blob containing the same texture across all FPs). Whenever DETE looks at a given blob with its EYE, it assigns to the EYE the same phase as that blob. This temporally based binding of attention makes sure that DETE only learns to associate verbal sequences with visual sequences of those blobs to which it

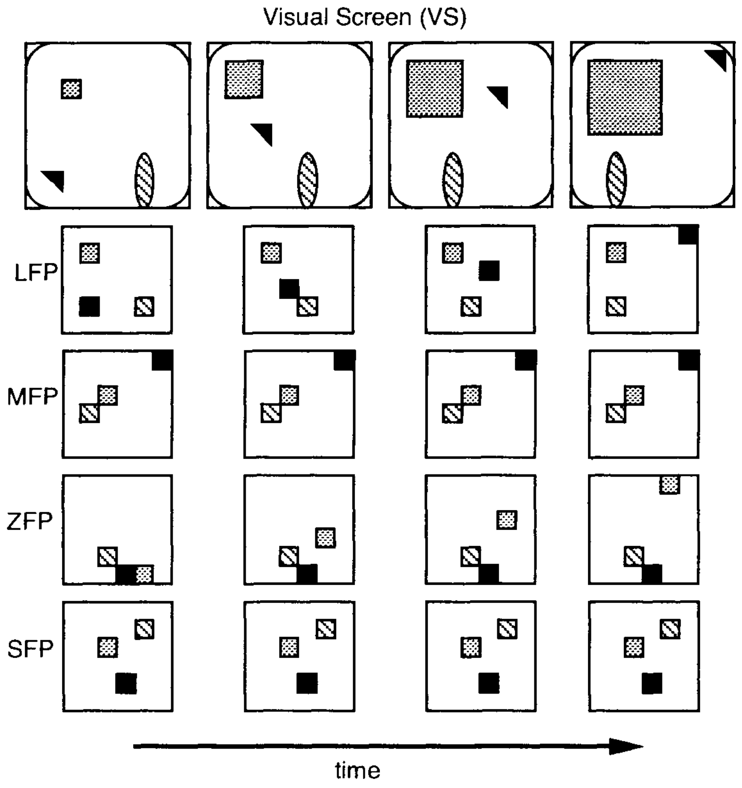


Figure 6 Visual Screen (VS) and Location (LFP), Motion (MFP), Size (ZFP) and Shape (SFP) Feature Planes (color, FINGER and EYE FPs are not shown here). Three blobs are moving on the VS. The oval blob is moving left; the square blob is growing and the triangular blob is moving diagonally upward toward the right. As blobs move/change on the VS, their active regions on the FPs are updated. Similar texture of active regions (small squares) indicates that these regions are firing in phase.

is attending. The use of multiple objects requires DETE to address the issue of attention. In contrast, Regier's model only contains 1 moving object, so it does not need to face this feature binding problem.

Feature Planes (FPs) are used as representational constructs for three reasons: (1) *Neuropsychological and Neurophysiological Support*: FPs correspond roughly to known neurophysiological and neuropsychological studies (Kandel, 1985) indicating both topographic mappings and that shape, position, etc. are processed in different regions of the brain and then reintegrated. (2) *Spatial Representational Analog*: Topographic layouts supply simplified, yet direct analogs for spatial features, and thus make representing space and motion easier. For example, a word like "up" can be represented by activity anywhere in the upper area of the Location Feature Plane. A word like "moves" be represented by activity anywhere away from the center of the MFP while directions of motion termed "diagonal" can be represented simply by activity anywhere in the diagonal regions of the MFP. FPs also support smooth generalization. If an object near the center of the MFP is moving slowly then objects mapped near to it will tend to be moving at about the same speed/direction. (3) *Combinatorial Learning and Generalization Capability*: Blob relationships and motions can be represented as a pattern of activity distributed over all FPs as they change sequentially in time. For example, the word "accelerate" can be represented and learned as a sequence of changing active regions, moving from the MFP's center toward its periphery. The use of separate/independent FPs also supports immediate generalization to novel combinations of known words. For instance, colors are mapped to one feature plane (i.e., CFP) while shapes are mapped to another (i.e., the SFP). As a result, once DETE has learned color terms and shapes (each separately), it can immediately understand novel combinations of these (e.g., "green ball," "green box," "red ball," etc.) — i.e., by activating an appropriate region of each distinct Feature Plane.

Each pattern sequence (i.e., of multiple active, changing regions over a given 2-D Feature Plane) is fed as input to a corresponding Feature Memory (FM). Each FM consists of a katamic memory composted of a 2-D array of novel neural elements, termed predictrons and recognitrons. Predictrons learn to predict the next input and recognitrons sample their neighbors' outputs. For each cell of a given Feature Plane there is a corresponding predictron/recognitron within the associated Feature Memory. Each predictron contains a linear sequence of dendritic compartments (DCs) in which information is propagated, in a pipeline fashion, toward the body (soma) of the predictron. Thus, each predictron acts as a temporal delay line. This shifting property is somewhat analogous to that of Time-Delay Neural Networks (TDNNs) (Waibel, 1989). In addition, each predictron samples information spatially (i.e., from neighboring predictrons) and temporally (i.e., from earlier stages in the pipeline of DCs of other predictrons). Katamic memory has both novel processing and learning capabilities. Figure 7 shows a simplified picture of katamic memory.

Experiments on katamic memory, reported in (Nenov, 1991; Nenov and Dyer in press-a, c), show that it has the following very useful properties: (1) *Rapid learning*: On average, only 4-6 exposures to a pattern sequence are sufficient for learning a novel sequence. This is 3-4 orders of magnitude improvement over recurrent PDP networks (Elman, 1990). (2) *Flexible memory capacity*: Multiple sequences of different lengths can be stored and the model is easily scalable to larger input patterns and/or sequences of greater length. (3) *Sequence completion/recall*: A short sequence (i.e., cue) is sufficient to discriminate and retrieve a previously recorded sequence. (4) *Fault and noise tolerance*: Missing bits can be tolerated and the memory can interpolate/extrapolate from existing data. (5) *Integrated learning and performance*: The katamic memory predictron can switch automatically from learning mode to performance mode. Thus, a katamic module can switch from learning to performance on a bit-by-bit and/or pattern-by-pattern basis. Also, whenever each predictron learns it uses positive evidence as a weak negative evidence for all other patterns. This allows DETE to learn, like Regier's model, from positive examples only.

10 FUTURE DIRECTIONS

The Regier and DETE systems are only first steps in grounding language learning in perceptual/motor experiences. Future directions in the area of representing perceptual experiences include: (a) extending the representations to 3-D objects, (b) representing composite objects with multiple motions — e.g., a set of hinged blobs could represent a boy versus a dog, or the more complex actions of eating (e.g., by movements of the lips with a shrinking blob in front of them) and (c) representing abstract concepts. At this point, no matter how much DETE sees one blob, say, attached to another, it will not really be learning the abstract concept of OWNERSHIP (since it is just seeing physical attachment). Almost by definition, abstractions are never observed directly and appear to have some innate basis in human brains (otherwise animals could learn such concepts as RESPONSIBILTY simply by observing irresponsible actions). Thus there is a need to build connectionist systems that unite perceptually-based language learning with the existence of mental states, such as plans, goals, themes and emotions (Dyer, 1983). At this point, only symbolic systems, e.g., (Dyer, 1983; Wilensky, 1983), are able to manipulate

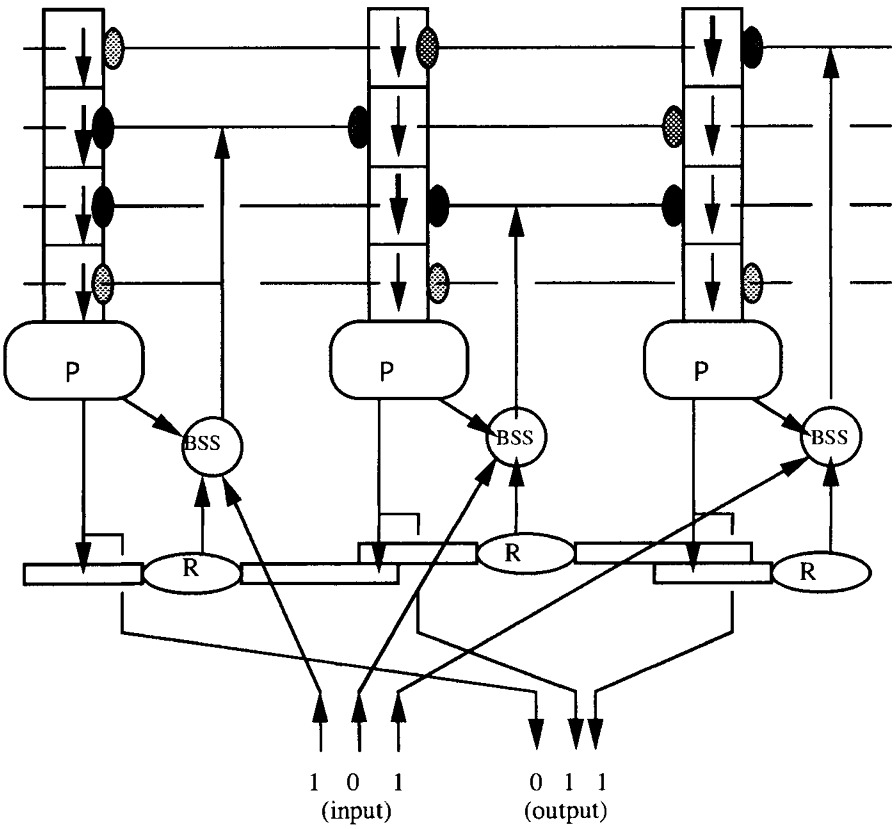


Figure 7 (Simplified) katamic memory, with only three predictrons (P), Recognitrons (R) and Bi-Stable Switches (BSS). Each BSS determines whether a predictron's Dendritic Compartments (DCs) get their input from the external environment or from internally generated outputs. Thus, BSSs are used for controlling when DETE attempts to perform sequence completion. Here, each predictron has only 4 DCs, illustrated as a train of squares above each predictron. Information about the input at a given region in a FP is shifted along the DCs in a pipeline fashion and is decayed over time (arrows of different thickness within each DC). Each recognitron here has two of its own dendritic compartments (RDCs), shown here as thin rectangles. These RDCs are used to sample the output of both its associated predictron and its neighbors (sampling of only one neighbor to-the-right shown here). Small vertical ovals indicate strength of input (via shading) to a predictronës DC, arriving from neighboring predictrons predictions (via lateral lines). In DETE, a typical katamic memory module will contain 256 predictrons with 64 DCs per predictron.

such constructs and in such systems these constructs are engineered by hand. The only distributed connectionist NLP system I am aware of that even attempts a goal/plan analysis of narrative text is the DYNASTY system (Lee, 1991; Lee and Dyer in press). DYNASTY's goal/plan analysis capabilities are extremely limited and one critical component (i.e., working memory) are implemented via symbol manipulation.

In general, we need only look at recent symbolic NLP models to find a wide variety of systems that exhibit different and important aspects of high-level reasoning — aspects not yet achieved by any connectionist model. For instance, the symbolic system OCCAM (Pazzani and Dyer, 1989; Pazzani, 1990) performs explanation-based learning (EBL) and thus can learn when given only a single example. EBL is not yet possible in connectionist models and Pazzani and Dyer (1987) have shown that backpropagation learning does concept formation in a way different than people (who have already built up some knowledge of the world). The symbolic story invention system MINSTREL (Turner 1992) not only creates new characters and events whose plot satisfies a theme, but it has both (a) analogical rules of invention, that map and adapt events from one domain to another, and (b) procedures that examine the appropriateness of a chain of events that have been created by a given heuristic rule. Within the connectionist paradigm, this chain-examination capability might be like having one network "examine" how well (or poorly) activation has spread along paths within another network. The symbolic system OpEd (Alvarado, 1990; Alvarado et al., 1990a, b, c) reads a fragment of editorial text and constructs an "argument graph" of beliefs (concerning the efficacy of plans) that are linked by attack/support relationships. The argument graph is then traversed to answer questions.

Barnden (1992a, b) points out the need for connectionist NLP systems that can build explicit representations of rules, e.g., as when one is asked to read a rule of the sort: "Any town that's been declared a disaster area [...] gets federal aid." followed by "Rotville has been declared a disaster area." (Barnden, 1992b, p. 29). Barnden's point is that we cannot rely on connectionist systems that just act as though they have rules (but without being able to access rules explicitly); otherwise we would not be able to build the rule and apply it, "on the fly," to conclude that Rotville will get disaster aid.

Another major direction for research is to understand how global CN structure might self organize — e.g., either through evolution (Werner and Dyer, 1991) or developmental self-organization (Kohonen, 1988). Currently, the connectionist knowledge engineer must, ahead of time, specify all of the major modules and ensembles and specify their paths of intermodule connectivity.

In the case of a single SRN there is little global structure to engineer. Unfortunately, SRNs have not been able to learn to perform the kinds of tasks achieved by CNs with multiple modules, such as DISCERN or DYNASTY. What the global structure of a connectionist network should be and how it might come about automatically is largely an open research issue.

11 CONCLUSIONS

What general morals can we take away with us, as the result of this overview? Here are a few: (a) There is no free lunch — clearly, complex knowledge-level architectures are needed — i.e., the mere existence of connectionist techniques is not going to eliminate the need for designing such architectures. General learning of complex cognitive tasks without preexisting network structure will always be intractable. Thus, some kind of "biasing" is required. NLP appears to be as complex as vision processing, so the structures needed may be as complex (or even more complex) as those in vision. (b) Time/space trade-offs will always exist — e.g., we see this with the trade-off between many fine-grain modules and a few coarse-grain modules. In general, more modules allow more pieces of knowledge to be manipulated in parallel. Architectures with fewer modules are easier to train but take longer to do so and end up being more sequential at the knowledge level. (c) Limited cognitive processing is acceptable if it is psychologically plausible — e.g., limits on the depth of a stack-like memory or on the number of identical predicates instances is reasonable if humans exhibit difficulty in processing similar texts. (d) Classical AI problems will remain for the foreseeable future — e.g., humans are able to both construct and apply rules on the fly. (d) Solving the "perceptually grounded language learning" problem will not, by itself, give us sophisticated NLP connectionist systems — abstractions must also somehow be encoded and/or acquired.

In spite of the difficulties facing connectionist NLP, it is still the case that great strides have been made. In the 1970s, NLP researchers built symbolic systems to read and answer questions about script-based stories. However, these systems were completely engineered. They did not learn any of their knowledge or processing skills. In the early 1990s, we see the arrival of, for instance, the DISCERN system — that can read and answer (simple) queries concerning restricted (i.e., single script) stories (Miikkulainen, 1993). However, it is important to realize that DISCERN acquires every piece of knowledge and every processing skill *through learning* — specifically, learning by example. DISCERN learns the meaning of words; it learns to parse word sequences into vectors representing case-role information; it learns to generate completed script event sequences; it learns to encode scriptal (and semantic role) information in its modules' weights; it learns to generate word sequences that describe events; it learns to parse questions and generate appropriate retrievals and it forms episodic memories though a process of self-organization. The only things that are engineered in DISCERN are: (a) the global form of the modules, (b) how information is routed from module to module during learning/performance, (c) the learning algorithm itself and (d) the set up and presentation of the training data. It is clear that DISCERN represents quite an accomplishment and has provided us with major insights into novel forms of representation and processing.

At one extreme there are connectionist researchers who believe that connectionist models will sweep away all forms of symbol manipulation (Churchland, 1986; Churchland and Sejnowski, 1989). At the other extreme are symbolically oriented researchers who claim that connectionism will never be more than a "mere implementation" (Pinker and Mehler, 1988) and "all of the action" is at the symbolic level. So far, the results are mixed. Connectionism has not advanced enough to offer alternative to the conveniences of symbol processing and thus attract away the majority of symbol pushers in traditional AI. If/when this happens, then connectionist processing will become preferred (since it offers a wide variety of nice features and potential links to brain research, etc.). However, existence of connectionist technologies and theories are not going to make knowledge representation, application of knowledge and reasoning issues magically disappear. Hopefully, connectionist technology and theory will continue developing and at some point the scales will tip in favor of connectionist implementations for all forms of high-level reasoning, but a need to understand processing at the knowledge/symbolic level will remain.

ACKNOWLEDGEMENTS

I would like to thank some of my past and current graduate students (S. Alvarado, C. Dolan, T. Lange, G. Lee, R. Miikkulainen, V. Nenov, M. Pazzani, R. Sumida, S. Turner, Y. Wang, and G. Werner) for their implementational and research contributions to NLP areas discussed here — areas spanning from symbolically based story invention to neurally and perceptually based language learning.

REFERENCES

[1] Ajjanagadde, V. and Shastri, L. (1989). Efficient Inference with Multi-Place Predicates and Variables in a Connectionist System. *Proceedings of the 11th Annual Conference of the Cognitive Science Society*. LEA Press, Hillsdale NJ. pp.396-403.

[2] Alvarado, S.J. (1990). *Understanding Editorial Text*. Kluwer, Norwell, MA.

[3] Alvarado, S., Dyer, M.G. and M. Flowers. (1990a). Argument Representation for Editorial Text. *Knowledge-Based Systems*, 3(2):87-107.

[4] Alvarado, S., Dyer, M.G. and M. Flowers. (1990b). Argument Comprehension and Retrieval for Editorial Text. *Knowledge-Based Systems*, 3(3).

[5] Alvarado, S.J., Dyer, M. G. and M. Flowers. (1990c). Natural Language Processing: Computer Comprehension of Editorial Text. In H. Adeli (Ed.), *Knowledge Engineering*, Vol. 1, Fundamentals. pp. 286-344, McGraw-Hill, NY.

[6] Barnden, J.A. (1991). Encoding complex symbolic data structures with some unusual connectionist techniques. In J.A. Barnden and J.B. Pollack (Eds.), *High-Level Connectionist Models*. Ablex, Norwood, NJ. pp. 180-240.

[7] Barnden, J.A. (1992a). Connectionism, Generalization and Propositional Attitudes: A Catalogue of Challenging Issues. In J. Dinsmore (Ed.), *The Symbolic and Connectionist Paradigms*: Closing the Gap. LEA Press, Hillsdale NJ. pp. 149-178.

[8] Barnden, J.A. (1992b). Connectionism, Structure-Sensitivity, and Systematicity: Refining the Task Requirements. *Memoranda in Computer and Cognitive Science*, No. MCCS-92-227, Computing Research Lab., New Mexico State University, Las Cruces NM.

[9] Barnden, J.A. (1994). Complex Symbol-Processing in a Transiently Localist Connectionist Architecture. This volume.

[10] Barnden, J. and Srinivas, K. (1991). Encoding Techniques for Complex Information Structures in Connectionist Systems. *Connection Science*, 3(3):269-315.

[11] Churchland, P.S. and T.J. Sejnowski. (1989). Neural Representation and Neural Computation. In L. Nadel, L.A. Cooper, P. Culicover and R.M. Harnish (Eds.), *Neural Connections, Mental Computation.* Bradford Book, MIT Press, Cambridge MA.

[12] Churchland, P.S. (1986). Neurophilosophy: *Toward a Unified Science for Mind-Brain.* MIT Press, Cambridge MA.

[13] Dolan, C.P. (1989). *Tensor manipulation networks: Connectionist and symbolic approaches to comprehension, learning, and planning.* Ph.d. Dissertation, Computer Science Dept. UCLA, Los Angeles, CA. (To be published by LEA Press).

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

[15] Dolan, C.P. and M.G. Dyer. (1989). Parallel Retrieval and Application of Conceptual Knowledge. In D. Touretzky, G. Hinton, T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, pp. 273-280.

[16] Dyer, M.G. (1983). *In-Depth Understanding.* MIT Press, Cambridge, MA.

[17] Dyer, M.G. (1990). Distributed Symbol Formation and Processing in Connectionist Networks. *Journal of Experimental and Theoretical Artificial Intelligence,* 2:215–239.

[18] Dyer, M.G., Cullingford, R. & Alvarado, S. (1987). Scripts. In S. Shapiro (Ed.), *Encyclopedia of Artificial Intelligence.* John Wiley & Sons, pp. 980-994.

[19] Dyer, M.G., Flowers, M. and Wang, Y.A. (1992). Distributed Symbol Discovery through Symbol Recirculation: Toward Natural Language Processing in Distributed Connectionist Networks. In R.G. Reilly and N.E. Sharkey (Eds.), *Connectionist Approaches to Natural Language Processing,* LEA Press, Hillsdale NJ, pp. 21-48.

[20] Dyer, M.G. and Nenov, V.I. (1993). Language Learning via Perceptual/Motor Experiences. *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society,* LEA Press, Hillsdale NJ.

[21] Elman, J.L. (1990). Finding structure in time. *Cognitive Science,* 14:179–211.

[22] Feldman, J.A. (1989). Neural Representation of Conceptual Knowledge. In L. Nadel, L.A. Cooper, P. Culiver and R.M. Harnish (Eds.), Neural Connections, Mental Computation. Bradford book/MIT PRess, Cambridge MA.

[23] Feldman, J.A., Lakoff, G., Stolcke, A. and Hollbach Weber, S. (1990). *Miniature Language Acquisition: A touchstone for cognitive science.* Tech. Rep. TR-90-009, ICSI, Berkeley, CA.

[24] Harnad, S. (1990). The Symbol Grounding Problem. *Physica D*, 42:335–346.

[25] Hinton, G.E., McClelland, J.L. and Rumelhart, D.E. (1986). Distributed Representation. In D.E. Rumelhart and J.L. Mcclelland (Eds.), *Parallel Distributed processing*, Vol. 1, MIT Press, Cambridge MA.

[26] Holldobler, S. (1990). A structured connectionist unification algorithm. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, Boston, MA.

[27] Johnson-Laird, P.N. (1983). *Mental Models: Towards a Cognitive Science of Language, Inference and Consciousness.* Harvard University Press, Cambridge MA.

[28] Kandel, E.R. (1985). Processing of Form and Movement in the Visual System. In E.R. Kandel and J.H. Schwartz (Eds.), *Principles of Neuroscience (Second Edition)*, (pp. 366-383), Elsevier, NY.

[29] Kohonen, T. (1988). *Self-Organization and Associative Memory.* Springer-Verlag (2nd ed.).

[30] Kolodner, J.L. (1984). *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model.* LEA Press, Hillsdale NJ.

[31] Lange, T.E. (1994). A structured connectionist approach to inferencing and retrieval. This volume.

[32] Lange, T.E. and M.G. Dyer. (1989a). Frame Selection in a Connectionist Model of High-Level Inferencing. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society (CogSci-89).* LEA Press, Hillsdale NJ.

[33] Lange, T. and M.G. Dyer. (1989b). Dynamic, Non-Local Role Bindings and Inferencing in a Localist Network for Natural Language Understanding. In D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 1.* San Mateo, CA: Morgan Kauffman, pp. 545-552,.

[34] Lange, T.E. and Dyer, M.G. (1989c). High-level inferencing in a connectionist network. *Connection Science*, 1:181–217.

[35] Lange, T.E., Vidal, J.J. and Dyer, M.G. (1991). Artificial Neural Oscillators for Inferencing. In A.V. Holden and V.I. Kryukov (Eds.), *Neurocomputers and Attention*, Vol. I, Manchester University Press.

[36] Lee, G.(1991). *Distributed Semantic Representations for Goal/Plan Analysis of Narratives in a Connectionist Architecture.* Ph.D. CS Dept. UCLA.

[37] Lee, G. and Dyer, M.G. (in press). Goal/Plan Analysis via Distributed Semantic Representations in a Connectionist System. *Applied Intelligence.*

[38] Lee, G., Flowers M. and M.G. Dyer. (1990). Learning Distributed Representations for Conceptual Knowledge and their Application to Script-Based Story Processing. *Connection Science*, 2(4):313–345. [Also reprinted in N. Sharkey (Ed.), Connectionist Natural Language Processing: *Readings from Connection Science.* (Chapter 11, pp. 215-247), Kluwer Academic Publishers, Norwell, MA. 1992.]

[39] Miikkulainen, R. and Dyer, M.G. (1991). Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*, 15(3):343–399.

[40] Miikkulainen, R. (1993). *Subsymbolic Natural Language Processing: An Integrated Model of Scripts, Lexicon and Memory*, Bradford/MIT Press, Cambridge MA.

[41] Miikkulainen, R. (1994). Subsymbolic Parsing of Embedded Structures. This volume.

[42] Minsky, M. (1985). *The Society of Mind.* Simon and Schuster, NY.

[43] Nenov, V.I. (1991). *Perceptually Grounded Language Acquisition: A Neural/Procedural Hybrid Model.* Ph.D. Dissertation and Technical Report UCLA-AI-91-07, Computer Science Departmentt, UCLA.

[44] Nenov, V.I. and Dyer, M.G. (in press-a) Perceptually Grounded Language Learning: Part 1 – A Neural Network Architecture for Robust Sequence Association. *Connection Science*, 5(2).

[45] Nenov, V.I. and Dyer, M.G. (in press-b) Perceptually Grounded Language Learning: Part 2 – DETE: A Neural/Procedural Model. *Connection Science*, 5(3).

[46] Nenov, V.I. and Dyer, M.G. (in press-c) Language Learning via Perceptual/Motor Association: A Massively Parallel Model. In Hiroaki Kitano (Ed.), *Massively Parallel Artificial Intelligence. AAAI/MIT Press.*

[47] Pazzani, M.J. (1990). *Creating a Memory of Causal Relationships: An Integration of Empirical and Explanation-Based Learning Method.* Lawrence Erlbaum Associates (LEA Press) Hillsdale, NJ, 1990.

[48] Pazzani, M.J. and M.G. Dyer. (1989). Memory Organization and Explanation-Based Learning. *International Journal of Expert Systems: Research & Applications*, 2(3):331-358.

[49] Pazzani, M. and M.G. Dyer. (1987). A Comparison of Concept Identification in Human Learning and Network Learning with the Generalized Delta Rule. *Proceedings of 10th Inaternational Joint Conference on Artificial Intelligence (IJCAI-87).* Morgan Kaufmann, Los Altos CA, pp. 147-150.

[50] Pinker, S. (1989). *Learnability and Cognition: The Acquisition of Argument Structure.* MIT Press, CAmbridge MA.

[51] Pinker, S. and Mehler, J. Eds., (1988). *Connections and Symbols.* Bradford/MIT Press. Cambridge, MA.

[52] Pollack, J.B. (1988). Recursive Auto-Associative Memory: Devising Compositional Distributed Representations. *Proceedings of the Tenth Annual Conference of the Cognitive Science Society* Lawrence Erlbaum. Hillsdale, NJ.

[53] Pollack, J.B. (1989). Implications of Recursive Distributed Representations. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing 1*, Morgan Kaufmann Publ. San Mateo, CA, pp. 527-536.

[54] Pollack, J.B. (1990). Recursive Distributed Representations. *Artificial Intelligence*, 46:77-105.

[55] Regier, T. (1992). *The Acquisition of Lexical Semantics for Spatial Terms: A Connectionist Model of Perceptual Categorization.* Ph.D. Dissertation. University of California at Berkeley.

[56] Rumelhart, D.E. and McClelland, J.L., Eds. (1986). *Parallel Distributed Processing.* Vol. 1. Bradford Books/MIT Press.

[57] St. John, M.F. and McClelland, J.L. (1990). Learning and Applying Contextual Constraints in Sentence Comprehension. *Artificial Intellgence*, 46:217-257.

[58] Schank, R.C. and Abelson, R.P. (1977). *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum, Hillsdale, NJ.

[59] Shastri, L. and Ajjanagadde, V. (1990). An Optimally Efficient Limited Inference System. *Proceedings of Eighth National Conference on Artificial Intelligence.* AAAI Press / MIT Press, Menlo Park, CA. pp. 563-570.

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

[61] Smolensky, P. (1990). Tensor Product Variable Binding and the Representation of Symbolic Structures in Connectionist Systems. *Artificial Intelligence,* 46:159–216.

[62] Stolcke, A. (1989). Unification as Constraint Satisfaction in Structured Connectionist Networks. *Neural Computation,* 1(4):559–567.

[63] Strong, G.W. and Whitehead, B.A. (1989). A solution to the tag-assignment problem for neural networks. *Behavioral and Brain Sciences,* 12:381–433.

[64] Sumida, R.A. and M.G. Dyer. (1989). Storing and Generalizing Multiple Instances while Maintaining Knowledge-Level Parallelism. *Proceedings of Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89).* pp. 1426-1431, (Distributed by Morgan Kaufmann Publ.).

[65] Sumida, R.A. (1991). Dynamic Inferencing in Parallel Distributed Semantic Networks. *Proceedings of Thirteenth Annual Conference of the Cognitive Science Society.* LEA Press, Hillsdale NJ. pp.913-917.

[66] Sumida, R.A. and Dyer, M.G. (1992). Propagation Filters in PDS Networks for Sequencing and Ambiguity Resolution. In J.E. Moody, S.J. Hanson and R.P. Lippmann (Eds.), *Advances in Neural Information Processing Systems 4,* Morgan Kaufmann Publ., San Mateo, CA, pp. 233-240.

[67] Sun, R. (1989). A discrete neural network model for conceptual representation and reasoning. *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society.* LEA Press, Hillsdale NJ.

[68] Sun, R. (1992). On Variable Binding in Connectionist Networks. *Connection Science,* 4(2):93–124.

[69] Sun, R. (1993). *Integrating Rules and Connectionism for Robust Reasoning.* John Wiley and Sons, NY.

[70] Touretzky, D.S. and G.E. Hinton. (1988). A Distributed Connectionist Production System. *Cognitive Science*, 12(3):423–466.

[71] Tulving, E. (1972). Episodic and Semantic Memory. In E. Tulving and W. D. Donaldson (Eds.), Organization of Memory. Academic Press, NY.

[72] Turner, S.R. (1992). MINSTREL: *A Computer Model of Creativity and Storytelling*. Ph.D. Dissertation, Computer Science Dept. UCLA (to be published by LEA Press).

[73] von der Malsburg, C. (1981). The correlation theory of brain function. Internal Report 81-2. Dept. of Neurobiology, Max-Plank-institute for Biophysical Chemistry.

[74] von der Malsburg, C. and Singer, W. (1988). Principles of cortical network organization. In P. Rakic and W. Singer (Eds.), *Neurobiology of Neocortex*. (pp. 69-99). John Wiley & Sons Ltd., London.

[75] Waibel, A. (1989). Consonant recognition by modular construction of large phonemic time-delay neural networks. In D.S. Touretzky (Ed.), *Advances in Neural Information Processing Systems I*. (pp. 215-223). Morgan Kaufman, San Mateo, CA.

[76] Werner, G. M. and M. G. Dyer. (1991). Evolution of Communication in Artificial Organisms. In J.D. Farmer, C. Langton, S. Rasmussen and C. Taylor (Eds.), *Artificial Life II*, Addison-Wesley.

[77] Wilensky, R. (1983). *Planning and Understanding: A Computational Approach to Human Reasoning*. Addison-Wesley, Reading, MA.

Appendix

Bibliography of Connectionist Models with Symbolic Processing

This bibliography was compiled by Ron Sun from contributions solicited from the research community. Especially valuable were the lists of references from Todd Lubert, Jim Garson, and Detlef Nauck. The selection of papers were based mainly on the following criterion: a paper x is included in this bibliography, **if and only if** x appears in a major conference, or x appears in a journal, or x appears in a well-edited book, or x is a monograph from a well-established publisher, or x is important and influential, **and** x is written in English. A deliberate effort was made to eliminate redundant or repetitive papers (but the results may still not be fully satisfactory).

The bibliography is annotated with the following categories:

- C — collections of papers.
- G — general discussions and surveys.
  For technical papers,
- A — analogical or case-based reasoning in neural networks.
- E — expert systems and neural networks.
- F — fuzzy logic and neural networks: hybrids and implementations.
- J — juxtaposition and linking of symbolic systems with neural networks.
- L — learning of symbolic structures.
- N — natural language, text, or speech processing.
- R — reasoning (rule-based and the like) and variable binding.
- S — schemas, scripts, semantic networks, or other conceptual structures.
  and
- O — topics other than the above.

Ajjanagadde, V. and Shastri, L. (1989). Efficient inference with multi-place predicates and variables in a connectionist system. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 396-403. Hillsdale, NJ: Erlbaum. [R]

Ajjanagadde, V. (1990). Reasoning with function symbols in a connectionist system. *Proceedings of the 12th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum. [R]

Anandan, P., Letovsky, S. and Mjolsness, E. (1989). Connectionist variable binding by optimization. *Proceedings of the 11th Cognitive Science Society*. [O]

Anderson, J. (1993). Data representation, neural networks, and hybrid computation. In Levine, D.S. and Aparicio, M. (eds), *Neural Networks for Knowledge Representation and Inference*. Hillsdale, NJ: Lawrence Erlbaum Associates. [G]

Ardizzone, E., Chella, A., Frixione, M., and Gaglio. S. (1992). Integrating subsymbolic and symbolic processing in artificial vision, *Journal of Intelligent Systems*, 1(4):273-308. [J]

Ballard, D. H. (1986). Parallel logical inference and energy minimization, In *Proceedings of the 5th National Conference on Artificial Intelligence*, Philadelphia, pp. 203-208. [R]

Barnden, J.A. (1984). On short-term information processing in connectionist theories. *Cognition and Brain Theory*, 7(1):25-59. [R]

Barnden, J.A. (1985). Diagrammatic short-term information processsing by neural mechanisms. *Cognition and Brain Theory*, 7(3 & 4):285-328. [R]

Barnden, J.A. (1988). The right of free association: relative-position encoding for connectionist data structures. In *Proc.of 10th Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Lawrence Erlbaum. pp. 503-509. [R]

Barnden, J.A. (1989). Neural-net implementation of complex symbol-processing in a mental model approach to syllogistic reasoning. In *Proc.of 11th Int. Joint Conf. on Artificial Intelligence* (Detroit, August 1989), pp. 568-573, San Mateo, CA: Morgan Kaufmann. [R]

Barnden, J.A. (1991). Encoding complex symbolic data structures with some unusual connectionist techniques. In Barnden, J.A. and Pollack, J.B. (eds.), *Advances in Connectionist and Neural Computation Theory 1: High-level connectionist models (Vol. 1)*, pp. 180–240 Ablex, Norwood, NJ. [R]

Barnden, J.A. (1992). Connectionism, generalization and propositional attitudes: a cAtalogue of challenging issues. In Dinsmore, J. (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap*, pp. 149–178, Hillsdale, NJ: Lawrence Erlbaum. [G]

Barnden, J.A, and Srinivas, K. (In Press). Overcoming rule-based rigidity and connectionist limitations through massively-parallel case-based reasoning. *Int. Journal of Man-Machine Studies.* [A]

Barnden, J.A. (in press). On the connectionist implementation of analogy and working memory matching. To appear in Holyoak, K.J. and Barnden, J.A. (eds), *Advances in Connectionist and Neural Computation Theory, Vol. 2: Analogical Connections.* Norwood, NJ: Ablex Publishing Corp. [A]

Barnden, J.A. (forthcoming). Connectionist meta-representation for propositional attitudes. *J. Experimental and Theoretical Artificial Intelligence*, 5(1). [R]

Barnden, J.A. and Pollack, J.B. (eds). (1991). *Advances in Connectionist and Neural Computation Theory, Vol. 1: High Level Connectionist Models.* Norwood, NJ: Ablex Publishing Corp. [C]

Barnden, J.A. and Srinivas, K. (1991). Encoding techniques for complex information structures in connectionist systems. *Connection Science*, 3(3):263–309. [R]

Bechtel, W. (1988). Connectionism and rules and representation systems: Are they compatible. *Philosophical Psychology*, 1(1):5-16. [R, G]

Becraft, W.R., Lee, P.L. and Newell, R.B. (1991). Integration of neural networks and expert systems for process fault diagnosis. In *Proc. 12th International Joint Conference on Artificial Intelligence*, pp. 832–837. [J, P]

Berenji, H.R. and Khedar, P. (1992). Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks*, 3(5):724–774. [F, L, C]

Bever, T.G. (1991). The demons and the beast — Modular and nodular kinds of knowledge. In Ronan, R. and Sharkey, N. (Eds.), *Connectionist Approaches To Natural Language Processing*. Lawrence Erlbaum (UK). [N, G]

Bezdek, J. (ed.) (1992). *IEEE Transaction on Neural Network*, special issue on fuzzy neural networks. [C, F]

Blank, D.S., Meeden, L.A., and Marshal, J.B. (1992). Symbolic manipulations via subsymbolic computations. In J. Dinsmore (ed.), *Closing the Gap: Symbolic vs. Subsymbolic Processing*, pp.113-149, Hillsdale, NJ: Lawrence Erlbaum. [G]

Bookman, L.A. (1989). A connectionist scheme for modeling context. In D. Touretzky et al. (eds.) In *Proc.1988 Connectionist Summer School*, pp.281-290. San Mateo, CA: Morgan Kaufmann. [N]

Bookman, L.A. (1991). Schema recognition for text understanding: An analog semantic feature approach. In Barnden, J.A. and Pollack, J.B. (Eds.), *Advances in Connectionist and Neural Computation Theory, (Vol.1)*. Norwood, NJ: Ablex. [N, J]

Bookman, L.A. (1993). A scalable architecture for integrating associative and semantic memory. *Connection Science*, 5(3&4):243–273. [J, L]

Bookman, L.A. (1994). *Trajectories Through Knowledge Space: A Dynamic Framework for Machine Comprehension*. Norwell, MA: Kluwer Academic Publishers. [J, L, N]

Bradshaw, G., Fozzard, R., and Ceci, L. (1989). A connectionist expert system that actually works. In *NIPS 88*, pp. 248–255. [E]

Brown, G. D. A. and Oaksford, M. (1990). The development of symbolic behaviour in natural and artificial neural networks. In Eckmiller, R., Hartmann, G., and Hauske, G. (eds), *Parallel Processing in Neural Systems and Computers*. Elsevier. [L]

Carpenter, G.A. et al. (1992). Fuzzy ARTMAP: A neural network architecture for incremental supervised learning of analog multidimensional maps. *IEEE Transactions on Neural Networks*, 3(5):698–713. [F]

Chalmers, D.J. (1990). Syntactic transformations on distributed representations. *Connection Science*, 2(1-2):53–62. [L]

Chalmers, D.J. (1990). Why Fodor and Pylyshyn were wrong: the simplest refutation. In *Proc.of the Twelfth Annual Conference of the Cognitive Science Society*. [L]

Chang, E. and Sekine, M. (1991). ARENA, a rule evaluating neural assistant that performs rule-based logic optimization. In *Proceedings of the International Joint Conference on Neural Networks*, pp.678–683, November 1991. [R]

Chorayan, O.G. (1982), Identifying elements of the probabilistic neuronal ensembles from the standpoint of fuzzy sets theory. *Fuzzy Sets and Systems*, 8(2):141–147. [F]

Chun, H. W. and Mimo, A. (1987). A model of schemata selection using marker passing and connectionist spreading activation. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*. pp. 887–896. Hillsdale, NJ: Erlbaum. [S]

Cooper, R. and Franks, B. (1991). Interruptibility: A new constraint on hybrid systems. *Artificial Intelligence and the Simulation of Behaviour Quarterly*. Special Issue on Hybrid Systems, 78:25–30. [G, J]

Cooper, R. and Franks, B. (1993). Interruptibility as a constraint on hybrid systems. *Minds and Machines*, 3(1):73–96. [G, J]

Cottrell, G. (1985). Connectionist parsing. In *Proceedings of the Seventh Annual Cognitive Science Society Conference*, Irvine, CA. [N]

Cottrell, G. (1985). Parallelism in inheritance hierarchies with exceptions. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Angeles, CA. Also in Al-Asady, R. and Narayanan, A. (Eds.), *Inheritance Networks for Artificial Intelligence*. Oxford: Intellect. [S]

Cottrell, G. and Fu-sheng Tsung (1991). Learning simple arithmetic procedures. In Barnden, J.A. and Pollack, J.B. (Eds.), *Advances in Connectionist and Neural Computation Theory, Vol 1: High-level Connectionist Models*, Norwood, NJ: Ablex. [L]

Cottrell, G.W. and Small, S.L. (1983). A connectionist scheme for modeling word sense disambiguation. *Cognition and Brain Theory*, 6(1):89–120. [N]

Das, S., Giles, C.L. and Sun, G.Z. (1992). Learning context free grammars: Capabilities and limitations of a neural network with an external stack memory. In *Proceedings of The Fourteenth Annual Conference of The Cognitive Science Society.* [L]

Dawes, R. (1993). Quantum neurodynamics and the representation of knowledge. In Levine, D.S. and Aparicio, M. (eds), *Neural Networks for Knowledge Representation and Inference.* Hillsdale, N.J.: Lawrence Erlbaum Associates. [G]

Derthick, M. (1990). Mundane reasoning by setting on a plausible model. *Artificial Intelligence*, 46(1-2):107–158, 1990. [R]

Derthick, M., and Plaut, D. C. (1986). Is distributed connectionism compatible with the physical symbol system hypothesis? In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* (pp. 639-644). Hillsdale, NJ: Erlbaum. [G]

Diamond, J., McLeod, R., and Pedrycz, W. (1990). A fuzzy cognitive system: examination of a referential neural architecture. In *International Joint Conference on Neural Networks 1990*, 2:617–622. [F]

Diederich, J. and Long, D.L. (1991). Efficient question answering in a hybrid system, In *Proceedings of the International Joint Conference on Neural Networks*, pp. 479–484, November 1991. [N]

Diederich, J. (1991). Steps towards knowledge-intensive connectionist learning. In Barnden J.A. and Pollack J.B. (eds.), *Advances in Connectionist and Neural Computation Theory*, Vol.1, pp. 284–303, Norwood, NJ: Ablex. [L]

Diederich, J. (1992). Explanation and artificial neural networks. *International Journal of Man-Machine Studies*, 37:335–355. [R]

Dinsmore, J. (Ed.). (1992). *Closing the Gap: Symbolism vs. Connectionism.* Lawrence Erlbaum Associates, Hillsdale, NJ. [C]

Dolan, C.P., and Dyer, M.G. (1987). Symbolic schemata, role binding and evolution of structure in connectionist memories. In *Proceedings of the First International Conference on Neural Networks.* [S, L]

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

Dolan, C. and M.G. Dyer. (1987). Towards the evolution of symbols. In *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications*. Cambridge, MA, July. [L]

Dolan, C.P. and Dyer, M.G. (1989). Parallel retrieval and application of conceptual knowledge. In Touretzky, D., Sejnowski, T.J. and Hinton, G. E (eds), *Proceedings of the 1988 Connectionist Models Summer School*, Morgan Kaufmann. [S]

Dyer, M.G. (1990). Distributed symbol formation and processing in connectionist networks. *Journal of Expt. Theor. Artificial Intelligence* , 2:215-239. [L]

Dyer, M.G. (1991). Symbolic neuroengineering for natural language processing: A Multiple level research approach. In Barnden, J.A. and Pollack, J.B. (Eds.), *Advances in Connectionist and Neural Computation Theory.* Vol. 1, NJ: Ablex. [G]

Dyer, M.G. (1991). Lexical acquisition through symbol recirculation in distributed connectionist networks. In Zernik, U. (Ed.), *Lexical Acquisition: Using On-Line Resources to Build a Lexicon.* Lawrence Erlbaum Assoc. Hillsdale, NJ, pp. 309-337. [N, L]

Elman, J. L. (1989). Structured representations and connectionist models. In *Proc.of Cognitive Science Conference*, pp. 17-23. [N, L]

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14:179-211. [N, L]

Eppler, W. (1990). Implementation of fuzzy production systems with neural networks. In Eckmiller, R., Hartmann, G., and Hauske, G. (eds.), *Parallel Processing in Neural Systems and Computers*, pp. 249-252 Elsevier. [F]

Fahlman, S. and Hinton, G. (1987). Connectionist architectures for artificial intelligence. *Computer*, 20:100-118. [G]

Farringdon. J. (1992). Approximating a neural network in symbolic logic. *Artificial Intelligence and the Simulation of Behaviour Quarterly*, Special Issue on Hybrid Systems, 79. [O]

Feldman, J. A., Fanty, M. A., Goddard, N. H., and Lynne, K. J. (1988). Computing with structured connectionist networks. *Communications of the ACM*, 31:170-187. [O]

Feldman, J. A. (1989). Neural representation of conceptual knowledge. In Nadel, Cooper, Culicover and Harnish (eds.), *Neural Connections, Mental Computation*, Cambridge MA: MIT Press. [S]

Fodor, J.A and Pylyshyn, Z.W. (1988). Connectionism and cognitive architecture: A critical analysis, *Cognition*, 28:3–71. Also in: Pinker and Mehler (eds.) *Connections and Symbols*, MIT Press, Cambridge, MA. [G]

Fu, L.M. (1991). Rule learning by searching on adapted nets. In *Proc.of AAAI'91*, pp. 590–595. [R, L]

Fu, L.M. (1990). Backpropagation in neural networks with fuzzy conjunction units. In *International Joint Conference on Neural Networks 1990*, 1:613–618. [F, L]

Fu, L.M. and Fu, L.C. (1990). Mapping rule-based systems into neural architecture. *Knowledge Based Systems*, 3:48–56. [R]

Fu, L.M. (1992). A Neural network model for learning rule-based systems. In *Proceedings of the International Joint Conference on Neural Networks*, I-343:I-348. [R, L]

Gallant, S. I. (1988). Connectionist expert systems. *Communications of the ACM*, 24(2):152–169. [R, E]

Gasser, M. and Dyer, M.G. (1988). Sequencing in a connectionist model of language processing. In *Proceedings of 12th International Conference on Computational Linguistics (COLING-88)*. Budapest, Hungary, August 1988. [N]

Giles, C.L., Sun, G.Z., Chen, H.H., Lee, Y.C. and Chen, D. (1990). High order recurrent networks and grammatical inference. In *Advances in Neural Information Processing System 2*, pp. 380–387, D.S. Touretzky (ed.), Morgan Kaufmann, San Mateo, CA. [L]

Giles, C.L., Chen, D., Miller, C.B., Chen, H.H., Sun, G.Z. and Lee, Y.C. (1991). Second-order recurrent neural networks for grammatical inference. In

Proceedings of International Joint Conference on Neural Networks, Vol. II, pp. 273–278, Seattle, Washington. [L]

Giles, C.L., Chen, D., Miller, C.B., Chen, H.H., Sun, G.Z. and Lee, Y.C. (1992). Extracting and learning an unknown grammar with recurrent neural networks. In Moody, J., Hanson, S., and Lippmann, R. (Eds.), *Advances in Neural Information Processing System 4*, pp. 17–324, Morgan Kaufmann, San Mateo, CA, 1992. [L]

Giles, C.L., Chen, D., Miller, C.B., Chen, H.H., Sun, G.Z. and Lee, Y.C. (1992). Learning and extractiing finite state automata with second-order recurrent neural networks. *Neural Computation*, Vol. 4, No. 3. [L]

Goebel, R. (1990). Learnign symbol processing with recurrent networks. In Eckmiller, R., Hartmann, G. and Hauske, G. (eds.), *Parallel Processing in Neural Systems and Computers*, pp. 157–160. Elsevier. [L]

Goebel, R. (1990). A connctionist approach to high-level cognitive modeling. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, Cambridge, MA. [O]

Goebel, R. (1990). Binding, episodic short-term memory, and selective attention, or why are PDP models poor at symbol manipulation? In Touretzky, D.S., Elman, J.L., Sejnowski, T.J. and Hinton, G.E. (Eds.), *Proceedings of the 1990 Connectionist Models Summer School.* San Mateo, CA: Morgan Kaufmann. [O]

Golden, R. M. (1986). Representing causal schemata in connectionist systems. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society.* pp. 13–21. Hillsdale, NJ: Erlbaum. [S]

Goodman, R.M. Miller, J. W. and Smyth, P. (1992). Rule-based neural networks for classification and probability estimation. *Neural Computation*, 4:781-804. [R]

Goodman, R. M., Miller, J. W. and Smyth, P. (1989). An information theoretic approach to rule-based connectionist expert systems. In *NIPS 88*, pp. 256-263. [R, E]

Greenspan, H.K. et al. (1992). Combined neural network and rule-based framework for probabilistic pattern recognition and discovery. In *Advances in Neural Information Processing Systems 4*, pp. 444–451. [R]

Grossberg, S. and Gutowski, W. (1987). Neural dynamics of decision making under risk: Affective balance and cognitive-emotional interactions. *Psychological Review*, 94(3):300–318. [R]

Güesgen, H. W. and Hölldobler, S. (1992). Connectionist inference systems. In Fronhfer, B. and Wrightson, G. (Eds.), *Parallelization in Inference Systems*. Springer, Lecture Notes in Artificial Intelligence, pp. 82–122. [R]

Gutknecht, M. and Pfeiffer, R. (1990). An approach to integrating expert systems with connectionist networks. *AI Communications*, 3(3):116–127. [E]

Hadley, R. F. (1990). Connectionism, rule following, and symbol manipulation. In *Proc. of AAAI-90*, pp. 579–586. [G, R]

Hall, L. O. and Romaniuk, S. G. (1990). FUZZNET: Towards a fuzzy connectionist expert system development tool. In *IJCNN 90*, Volume 2, pp. 483-486, Washington, DC. [E]

Handelman, D.A., Lane, S.H., and Gelfand, J.J. (1990). Integrating knowledge-based system and neural network techniques for robotic skill acquisition. In *International Joint Conference on Neural Networks*, 193–198. Also in: *Proceedings of IJCAI*, pp. 193–198. [E, J]

Harnad, S. (1990). The symbol grounding problem. *Physica D*, 42(1-3):335–346. [J]

Harnad, S. (1992). Connecting object to symbol in modeling cognition. In Clark, A. and Lutz, R. (eds.), *Connectionism in Context*. Springer. [J]

Hawthorne, J. (1989). On the compatibility of connectionist and classical models. *Philosophical Psychology*, 2(1):5–15. [G]

Harris, C.L. and Elman, J.L. (1989). Representing variable information with simple recurrent networks. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. pp. 635–642. Hillsdale, NJ: Erlbaum. [O]

Hayashi, I., Nomura, H., Yamasaki, H. and Wakami, N. (1992). Construction of fuzzy inference rules by neural network driven fuzzy reasoning and neural network driven fuzzy reasoning with learning functions. *International Journal of Approximate Reasoning*, 6(2):241-266. [F, L]

Hayashi, Y., Krishnamraju, P.V., and Reilly, K.D. (1991). An architecture for hybrid expert systems. In *Proc. of Int'l Joint Conf. on Neural Networks* (IJCNN'91-Singapore), Nov. 18-21, pp. 2773-2778. [E]

Henderson, J. (1992). A connectionist parser for structure unification grammar. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics,* Newark, DE. [N]

Hendler, J. (1991). Developing hybrid symbolic/connectionist models. In Barnden J.A. and Pollack J.B. (eds.), *Advances in Connectionist and Neural Computation Theory,* pp. 165–179. Hillsdale, NJ: Lawrence Erlbaum Assoc. [J]

J. Hendler, (1987). Marker passing and microfeatures. In *Proc.10th IJCAI,* pp. 151–154, Morgan Kaufman, San Mateo, CA. [J]

Hinton, G. E. (ed.). (1990). Special Issue on Connectionist Symbol Processing. *Artificial Intelligence,* 46. Also as *Connectionist Symbol Processing.* Cambridge, MA: Bradford Books/MIT Press. [C]

Hinton, G. E. (1990). Mapping part-whole hierarchies into connectionist networks. *Artificial Intelligence,* 46:47–76. [S]

Hirota, K. and Pedrycz, W. (1991). Fuzzy logic neural networks: Design and computations. In *Proceedings of the International Joint Conference on Neural Networks,* pp. 154–157. [F]

Horgan, T. and J. Tienson, J. (in press). Structured representations in connectionist systems? In Davis, S. (ed.), *Connectionism: Theory and Practice.* British Columbia. [G]

Hollatz, J. (1992). Supplementing neural network learning with rule-based knowledge. In *Proceedings of International Joint Conference on Neural Networks (IJCNN 92),* Beijing, III-59–III-600. [L, R]

Hölldobler, S. (1990). A structured connectionist unification algorithm. In *Proc.of AAAI'90,* pp. 587–593. [O]

Hölldobler, S. and Kurfeß, F. (1991). CHCL — A connectionist inference system. In Fronhöfer, B. and Wrightson, G. (eds.), *Parallelization in Inference Systems,* Lecture Notes in Computer Science. Springer. [R]

Holyoak, K.J. and Barnden, J.A. (eds). (in press). *Advances in Connectionist and Neural Computation Theory, Vol. 2: Analogical Connections.* Norwood, NJ: Ablex Publishing Corp. [C]

Honavar, V. and Uhr, L. (1993). Generative learning structures and processes for generalized connectionist networks. *Information Sciences.* Special Issue on Neural Networks and Artificial Intelligence. [L]

Honavar, V. and Uhr, L. (Eds). (in press). *Symbol processors and connectionist networks in artificial intelligence and cognitive modelling.* New York: Academic Press. [C]

Honavar, V. (in press). Connectionist learning with structured symbolic representations. In Honavar, V. and Uhr, L. (Eds.), *Symbol Processors and Connectionist Network Models in Artificial Intelligence and Cognitive Modelling.* New York: Academic Press. [L]

Jagota, A. (1993). Representing discrete structures in a Hopfield-style network. In Levine, D.S. and M. Aparicio (eds.), *Neural Networks in Knowledge Representation and Inference*, Associates. [S]

Jang, J-S. R. (1992). Self-learning fuzzy controllers based on temporal back propagation. *IEEE Transactions on Neural Networks*, 3(5): 714–723. [F, L]

Jang, J-S. R. (1991). Fuzzy modeling using generalized neural networks and kalman filter algorithm. In *Proc. of the Ninth National Conference on Artificial Intelligence (AAAI-91)*, pp. 762–767, [F]

Jang, J-S. R. (1992). ANFIS: Adaptive-network-based fuzzy inference systems. *IEEE Transaction on System, Man, and Cybernetics*, 23:3. [F, L]

Jones, M.A. (1987). Feedback as a coindexing mechanism in connectionist architectures. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI)*, Milan, Italy, August 1987. [N]

Jones, M.A. and Driscoll, A.S. (1985). Movement in active production networks. In *Proceedings of the 23rd Annual Meeting of the Association for Computational Linguistics*, Chicago, July, pp. 161–166. [N]

Kanal, L. and Raghavan, S. (1992). Hybrid systems: A key to intelligent pattern recognition. In *Proceedings of the International Joint Conference on Neural Networks*, IV:177–183. [J]

Keller, J.M. and Tahani, H. (1992). Implementation of conjunctive and disjunctive fuzzy logic rules with neural networks. *International Journal of Approximate Reasoning*, 6(2):221-240. [F]

Keller, J.M. and Yager, R.R. and Tahani, H., (1992). Neural network implementation of fuzzy logic. *Fuzzy Sets and Systems*, 45:1-12. [F]

Keller, J.M. et al. (1992). Evidence aggregation networks for fuzzy logic inference. *IEEE Transactions on Neural Networks*, 3(5):761-769. [F]

Kentridge, R.W. (in press) Cognition, chaos and non-deterministic symbolic computation: The Chinese room problem solved? To appear in *Think* (special issue on Connectionism and Symbolic Artificial Intelligence). [G]

Kosko, B. (1986). Fuzzy cognitive maps. *Int. J. Man-Machine Studies*, 24:65-75. [F]

Kuncicky, D.C. (1989). A fuzzy interpretation of neural networks. *The 3rd IFSA Congress*, pp. 113-116. [F]

Kuncicky, D. C., Hruska, S. I., and Lacher, R. C. (1991). Hybrid systems: The equivalence of rule-based expert system and artificial neural network inference. *International Journal of Expert Systems*. [E]

Kurfeß, F. (1991). Unification on a connectionist simulator. In *International Conference on Artificial Neural Networks ICANN-91*, Helsinki, Finland. [O]

Kwasny, S.C. and Faisal, K.A. (1990). Connectionism and determinism in a syntactic Parser. *Connection Science*. 2(1-2):63-82. [N]

Lacher, R. C., Hruska, S. I., and Kuncicky, D. C. (1991). Backpropagation learning in expert networks. *IEEE Transactions on Neural Networks*. [E, L]

Lacher, R.C. (1993). Expert networks: Paradigmatic conflict, technological rapprochement. *Minds and Machines*, 3:53-71. [E]

Lachter, J. and Bever, T.G. (1988). The relation between linguistic structure and associative theories of language learning—A constructive critique of some connectionist learning models. *Cognition*, 28:195-247. [N]

Lange, T. E. and Dyer, M. G. (1989). Frame selection in a connectionist model of high-level inferencing. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 706–713. Hillsdale, NJ: Erlbaum. [R]

Lange, T. E. and Dyer, M. G. (1989). High-level inferencing in a connectionist network. *Connection Science*, 1:181–217. [R]

Lange, T.E. (1990). Analogical retrieval within a hybrid spreading-activation network. In *Proceedings of the 1990 Connectionist Models Summer School*, pp. 265–274, 1990. [A]

Lange, T.E. and Wharton C. (in press). REMIND: Retrieval from episodic memory by inferencing and disambiguation. To appear in Barnden, J.A. and Holyoak, K. (Eds.), *Advances in Connectionist and Neural Computation Theory, Volume 2: Analogical Connections*. Norwood, NJ: Ablex. [A]

Lea, R.N. and Villareal, J. (eds.) (1991). *Proceedings of the Second Joint Technology Workshop on Neural Networks and Fuzzy Logic*. NASA Lyndon B. Johnson Space Center, Houston, Texas. [C, F]

Lee, C.C. (1990). A self-learning rule-based controller employing approximate reasoning and neural network concepts. *Int. J. Intell. Syst.*, 5(3). [F]

Lee, S. and Lee, E. (1974). Fuzzy sets and neural networks. *J. Cybern.*, 4(2):83–103. [F]

Lee, G., Flowers, M., and Dyer, M.G. (1989). A symbolic/connectionist script applier mechanism. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. pp. 714–721. Hillsdale, NJ: Erlbaum. [S]

Lee, G., Flowers, M., and Dyer, M.G. (1990). Learning distributed representations of conceptual knowledge and their application to script based story processing. *Connection Science*, 2(4):313–346. [S]

Lehnert, W.G. (1991). Symbolic/subsymbolic sentence analysis: Exploiting the best of both worlds. In Barnden, J.A. and Pollack, J.B. (Eds.), *Advances in Connectionist and Neural Computation Theory*. Hillsdale, NJ: Lawrence Erlbaum Assoc. [N]

Levine, D.S. and Aparicio, M. (Eds). (1993). *Neural Networks for Knowledge Representation and Inference*. Hillsdale, NJ: Lawrence Erlbaum Associates. [C]

Lim, J.H. et al. (1991). INSIDE: A connectionist case-based diagnostic expert system that learns incrementally. In *Proceedings of the International Joint Conference on Neural Networks*, pp. 1693–1698. [A, E]

Lim, J, Lui, H., and Wang, P., (1992). A framework for integrating fault diagnosis and increamental knowledge acquisition in connectionist expert systems. In *Proc.of AAAI'92*, pp. 159–165. [L, E]

Lin, C.-T. and Lee, C. S. G. (1991). Neural-network-based fuzzy logic control and decision system. *IEEE Transactions on Computers — Special Issue on Artificial Neural Networks*, 40(12):1320–1336. [F]

Machado, R.J. and Rocha, A.F. (1992). A hybrid architecture for fuzzy connectionist expert systems. In Kandel, A. and Langholz, G. (Eds.), *Hybrid Architectures for Intelligent Systems*. CRC Press Inc. [F, E]

Machado, R.J. and Rocha, A.F. (1992). Evolutive fuzzy neural networks. In *Proceedings of the IEEE International Conference on Fuzzy Systems*, San Diego, CA. [F, L]

MacLennan, B. J. (in press). Characteristics of connectionist knowledge representation. *Information Sciences*. [O]

MacLennan, B. J. (1993). Continuous symbol systems: The logic of connectionism. In Levine, D.S. and Aparicio IV, M. (eds.), *Neural Networks for Knowledge Representation and Inference*. Hillsdale NJ: Lawrence Erlbaum. [O]

MacLennan, B. J. (in press). Image and symbol: Continuous computation and the emergence of the discrete. In Honavar, V. and Uhr, L. (eds.), *Integrating Symbolic Processors and Connectionist Networks for Artificial Intelligence and Cognitive Modelling*. New York, NY: Academic Press. [O]

Maclin, R. and Shavlik, J. (1993). Using knowledge-based neural networks to improve algorithms: Refining the Chou-Fasman algorithm for protein folding. *Machine Learning*, 11:195–215. [R, L]

Mangold-Allwinn, R. (1990). Learning to produce discriminative object descriptions: On the representation of rules in a PDP net. In Eckmiller, R., Hartmann, G., and Hauske, G. (eds.), *Parallel Processing in Neural Systems and Computers*, pp. 487–490. Elsevier. [R]

Mani, D.R. and Shastri, L. (1991). Combining a Connectionist Type Hierarchy with a Connectionist Rule-Based Reasoner. In *Proceedings of the Thirteenth Conference of the Cognitive Science Society*, pp. 418–423, Chicago, IL. [R, S]

McClelland, J.L. and Kawamoto, A.H. (1986). Mechanisms of sentence processing: assigning roles to constituents. In McClelland, J.L. and Rumelhart, D.E. (Eds.), *Parallel Distributed Processing: Explorations in The Microstructure of Cognition I.* MIT Press, Cambridge, MA. [N, S]

McMillan, C. et al. (1992). Rule induction through integrated symbolic and subsymbolic processing. In *Advances in Neural Information Processing Systems 4*, pp. 969–976. [L, R]

McMillan, C. and Smolensky, P. (1988). Analyzing a connectionist model as a system of soft rules. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.* pp. 62–68. Hillsdale, NJ: Erlbaum. [R]

Miikkulainen, R. and Dyer, M.G. (1988). Forming global representations with extended backpropagation. In *Proceedings of the IEEE Second Annual International Conference on Neural Networks (ICNN-88).* San Diego, CA, July. [L]

Miikkulainen, R. and Dyer, M. G. (1989). A modular neural network architecture for sequential paraphrasing of script-based stories. In *Proc. of the Second Joint Conference on Neural Networks*, pp. 29–56, Washington D.C. [N, S]

Miikkulainen, R. and Dyer, M.G. (1989). Encoding input/output representations in connectionist cognitive systems. In Touretzky, D.S., Hinton, G. and Sejnowski, T. (eds.), In *Proceedings of the 1988 Connectionist Models Summer School.* Morgan Kaufmann, San Mateo, CA. [O]

Miikkulainen, R. and Dyer, M.G. (1991). Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science.* 15(3). [N, S]

Minsky, M. (1990). Logical vs. analogical or symbolic vs. connectionist or neat vs. scruffy. In *Frontiers of Artificial Intelligence*, Chapter 9, pp. 218–243. MIT Press. [G]

Mitchell, M. and Hofstadter, D.R. (in press). The emergence of understanding in a computer model of concepts and analogy-making. *Physica D.* [A]

Mitchell, M., and Hofstadter, D.R. (1989). The role of computational temperature in a computer model of concepts and analogy-making. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society* (pp. 765-772). Hillsdale, NJ: Erlbaum. [A]

Moisl, H. (1992). Connectionist finite state natural language processing, *Connection Science.* 4(2):67-91. [N]

Montgomery, G.J. and Drake, K.C. (1991). Abductive reasoning networks. *Neurocomputing*, 2(3):97-104. [R]

Myllymäki, P., Tirri, H., Floréen, P. and Orponen, P. (1990). Compiling high-level specifications into neural networks. In *Proceedings of the International Joint Conference on Neural Networks* (Washington D.C., January 1990), Vol 2: 475-478. IEEE, New York, NY. [O]

Myllymäki, P. and Tirri, H. (1993). Bayesian case-based reasoning with neural networks. In *Proceedings of the IEEE International Conference on Neural Networks*, San Francisco. [A]

Nauck, D. and Kruse, R. (1993). A fuzzy neural network learning fuzzy control rules and membership functions by fuzzy error backpropagation. In *Proc. IEEE Int. Conf. on Neural Networks 1993*, San Francisco, pp. 1022-1027. [F]

Oden, G. C. (1988). FuzzyProp: A symbolic superstrate for connectionist models. In *Proceedings of the IEEE International Conference on Neural Networks*, Vol. I, 293-300. [F]

Oden, G. C. (1992). Direct, incremental learning of fuzzy propositions. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 48-53. [F, L]

Oden, G. C. (in press). Why the difference between connectionism and anything else is more than you might think but less than you might hope. In Honavar, V. and Uhr, L. (eds.), *Integrating Symbol Processors and Connectionist Networks in Artificial Intelligence and Cognitive Modeling.* New York, NY: Academic Press. [F]

Oliver, W.L. and Schneider, W. (1988). Using rules and task division to augment connectionist learning. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society.* pp. 55-61. Hillsdale, NJ: Erlbaum. [R, L]

Omlin, C.W. and Giles, C.L. (1992). Training second-order recurrent neural networks using hints. In *Proceedings of the Ninth International Conference on Machine Learning*, pp. 363–368, Morgan Kaufmann, San Mateo, CA. [L]

Omlin, C.W., Giles, C.L., and Miller, C.B. (1992). Heuristics for the extraction of rules from discrete-time recurrent neural networks. In *Proceedings International Joint Conference on Neural Networks*, I, 33–38. [R, L]

Opitz, D.W. and Shavlik, J.W. (1993). Heuristically expanding knowledge-based neural networks. In *Proc. of the 1993 International Joint Conference on Artificial Intelligence.* [R]

Orponen, P., Floréen, P. Myllymäki, P., and Tirri, H. (1990). A neural implementation of conceptual hierarchies with bayesian reasoning. In *Proceedings of the International Joint Conference on Neural Networks* (San Diego, CA, June 1990), Vol 1: 297–303. IEEE, New York. [S]

Peng, Y. and Reggia, J.A. (1989). A connectionist model for diagnostic problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(2):285–298. [E]

Pinkas, G. (1991). Energy minimization and the satisfiability of propositional calculus. *Neural Computation*, 3(2):282–291. Also in Touretzky, D.S., Elman, J.L., Sejnowski, T.J. and Hinton, G.E. (eds), *Proceedings of the 1990 Connectionist Models Summer School*, pp. 23–31, San Mateo, Morgan Kaufmann. [R]

Pinkas, G. (1991). Propositional non-monotonic reasoning and inconsistency in symmetric neural networks. In *International Joint Conference on Artificial Intelligence (IJCAI-91).* pp. 525–530. [R]

Pinkas, G. (1992). Constructing proofs in symmetric networks. In Moody, J.E, Hanson, S.J., and Lipmann, R.P. (eds.), In *Advances in Information Processing Systems 4* (NIPS), pp. 217–224. [R]

Pinkas, G. and Dechter, R. (1992). A new improved activation function for connectionist energy minimization. In *Proceedings of The Tenth National Conference on Artificial Intelligence*, pp. 434–439, San Jose, CA. [R]

Pinker, S. and Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language inquisition. *Cognition.* 28:73–193. [N]

Plate, T. A. (1991). Holographic reduced representations: Convolution algebra for compositional distributed representations. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, pp. 30–35, Sydney, Australia. Also in: *Artificial Neural Networks: Concepts and Theory*, IEEE Computer Society Press Tutorial, Mehra, P. and Wah, B.W. (eds.), pp. 126–131, Los Alamitos, CA. [O]

Plate, T.A. (1992). Holographic recurrent networks. In Giles, C.L., Hanson, S.J. and Cowan, J.D. (Eds.), *Advances in Neural Information Processing Systems 5* (NIPS'92), Morgan Kaufmann, San Mateo, CA. [O]

Pollack, J.B. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *Proc.of 10th Cognitive Science Society Conference.* [O]

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, 46:77–105. [O]

Riley, M. S. and Smolensky, P. (1984). A parallel model of (sequential) problem solving. In *Proceedings of the Sixth Annual Conference of the Cognitive Science Society.* Hillsdale, NJ: Erlbaum. [R]

Robins, A.V. (1992). Multiple representations in connectionist systems. *International Journal of Neural Systems*, 2(4):345–362. [O]

Rose, D.E and Belew, R.K (1989). A case for symbolic/sub-symbolic hybrids. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, MI, pp. 844–851. [J]

Rose, D.E and Belew, R.K (1991). A connectionist and symbolic hybrid for improving legal research. *International Journal of Man-Machine Studies*, 35:1–33. [J]

Rose, D.E. (1990). Appropriate uses of hybrid systems. In Touretzky, D.S., Elman, J.L., Sejnowski, T.J. and Hinton, G.E. (eds), *Proceedings of the 1990 Connectionist Models Summer School*, Morgan Kaufmann, San Mateo, CA, pp. 277–286. [J]

Rumelhart, D.E. (1989). Toward a microstructural account of human reasoning. In Vosniadou, S. and Ortony, A. (eds.), *Similarity and Analogical Reasoning*, (pp. 298–312). New York: Cambridge University Press. [A]

Rumelhart, D.E., Smolensky, P., McClelland, J.L., and Hinton, G.E. (1986). Schemata and sequential thought processes in PDP models. In McClelland, J.L., Rumelhart, D.E. and The PDP Research Group (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, (Vol. 2, pp. 7–57). Cambridge, MA: Bradford Books. [S]

Rumelhart, D.E. and Todd, P.M. (in press). Learning and connectionist representations. In Meyers, D. and Kornblum, S. (Eds.), *Attention and Performance*, Cambridge, MA: MIT Press. [L]

Samad, T. (1988). Towards connectionist rule-based systems. *IEEE International Conference on Neural Networks*, Vol.2, pp. 525–532. [R]

Schneider, W. and Oliver, W. L. (in press). An instructable connectionist/control architecture: Using rule-based instructions to accomplish connectionist learning in a human time scale. In VanLehn, K. (Ed.), *Architectures for Intelligence*. Hillsdale, NJ: Erlbaum. [R]

Sharkey, N. E. (1991). Connectionist representation techniques. *AI Review*, (5):142–167. [O]

Sharkey, N.E. (1992). The ghost in the hybrid: A study of uniquely connectionist representations. *AISB Quarterly*, 10–16. [O]

Sharkey, N.E. and Sutcliffe, R.F.E. and Wobcke, W.R. (1986). Mixing binary and continuous connection schemes for knowledge access. In *Proceedings of the American Association for Artificial Intelligence*, pp. 262–266. [O]

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

Shastri, L. (1988). A connectionist approach to knowledge representation and limited inference. *Cognitive Science*, 12:331–392. [R]

Shastri, L. (1988). *Semantic Networks: An Evidential Formulation and its Connectionist Realization*. Pitman, London, UK. [S]

Shastri, L. and Feldman, J. A. (1985). Evidential reasoning in semantic networks: A formal theory. In *Proc.of IJCAI'85*, pp. 465–474. [S]

Shavlik, J.W. and Towell, G.G. (1989). An approach to combining explanation-based and neural learning algorithms. *Connection Science*, 1(3):233–255. [L]

Simpson, P.K. 1992. Fuzzy min-max neural networks: 1. Classification. *IEEE Transaction on Neural Networks.* [F]

Simpson, P.K. 1992. Fuzzy min-max neural networks: 2. Clustering. *IEEE Transaction on Fuzzy Systems.* [F]

Sloman, S.A. (in press). Feature-based induction. *Cognitive Psychology.* [O]

Small, S.L., Cottrell, G.W., and Shastri, L. (1982). Toward connectionist parsing. *Proceedings of the National Conference on Artificial Intelligence*, Pittsburgh, PA. [N]

Smolensky, P. (1987). On the connectionist reduction of conscious rule interpretation. In *Proceedings of the Ninth Annual Conference of the Cognitive Science Society.* pp. 187–194. Hillsdale, NJ: Erlbaum. [R]

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

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

Smolensky, P. (1991). Connectionism, constituency and the language of thought. In Loewer, B. and Rey, G. (Eds.), *Fodor and his Critics.* Blackwell, Oxford, UK. [G]

Sohn, A. and Gaudiot, J.L. (1991). Connectionist production systems in local and hierarchical representation. In Bourbakis, N.G. (Ed.), *Applications of Learning and Planning Methods*, pp. 165–180, World Scientific Publishing. [R]

Soucek, B. and The IRIS Group (Eds.) (1991). *Neural and Intelligent Systems Integration.* Wiley, New York, NY. [C]

St. John, M. F. and McClelland, J. L. (1991). Learning and applying contextual constrains in sentence comprehension. *Artificial Intelligence*, 46(1-2):217-257. [N]

Stolcke, A. (1989). Unification as constraint satisfaction in structured connectionist networks, *Neural Computation*, 1. [O]

Stucki, D. J. and Pollack, J. B. (1992). Fractal (reconstructive analogue) memory. In *14th Annual Conference of the Cognitive Science Society*, Bloomington, IN. [O]

Sumida, R.A., Dyer, M.G., and Flowers, M. (1988). Integrating marker passing and connectionism for handling conceptual and structural ambiguities. In *Proceedings of the 10th conf of the Cognitive Science Society*, Montreal. [N]

Sumida, R.A. and Dyer, M.G. (1989). Storing and generalizing multiple instances while maintaining knowledge-level parallelism. In *Proceedings of Eleventh International Joint Conference on Artificial Intelligence* (IJCAI-89). pp. 1426–1431. Morgan Kaufmann Publ. San Mateo, CA. [S]

Sumida, R.A. and Dyer, M.G. (1992). Propagation filters in PDS networks for sequencing and ambiguity resolution. In Moody, J.E., Hanson, S.J., and Lippmann, R.P. (eds.), *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann, San Mateo, CA, pp. 233–240. [S]

Sun, R. (1989). A discrete neural network model for conceptual representation and reasoning. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum. [R]

Sun, R. (1991). Connectionist models of rule-based reasoning. In *Proc.13th Cognitive Science Conference*, pp. 437–442, Lawrence Erlbaum Associates, Hillsdale, NJ. [R, J]

Sun, R. (1991). The discrete neuronal model and the probabilistic discrete neuronal model. In Soucek, B. (ed.), *Neural and Intelligent Systems Integration*, pp. 161–178 John Wiley and Sons, New York, NY. [L]

Sun, R. (1992). A connectionist model for commonsense reasoning incorporating rules and similarities. *Knowledge Acquisition*, 4:293–321. [R, J]

Sun, R. (1992). Fuzzy Evidential logic: A model of causality for commonsense reasoning. In *Proc.14th Cognitive Science Society Conference*, pp. 1134–1139, Lawrence Erlbaum Associates, Hillsdale, NJ. [R]

Sun, R. (1992). On variable binding in connectionist networks. *Connection Science*, 4(2):93–124. [R]

Sun, R. (1993). Beyond associative memories: Logics and variables in connectionist networks. *Information Sciences*, special issue on AI and neural networks, 70(1,2). [R]

Sun, R. (1993). An efficient feature-based connectionist inheritance scheme. *IEEE Transaction on System, Man and Cybernetics*, 23(2):1–12. [S]

Sun, R. (1993). A neural network model of causality. *IEEE Transaction on Neural Networks*. [R]

Sun, R. (1993). *Integrating Rules and Connectionism for Robust Commonsense Reasoning*. John Wiley and Sons, New York, NY. [G, R, S]

Sun, R. (in press). Structuring knowledge in vague domains. *IEEE Transaction on Knowledge and Data Engineering*. [R, J]

Sun, R. and Bookman, L.A. (1993). How do symbols and networks fit together? *Artificial Intelligence* magazine. [G]

Sun, R., Bookman, L., and Shekhar, S. (eds.) (1992). *The Working Notes of the AAAI Workshop on Integrating Neural and Symbolic Processes*. American Association for Artificial Intelligence, Menlo Park, CA. [C]

Sun, R. and Waltz, D.L. (1991). A neurally inspired massively parallel model of rule based reasoning. In Soucek, B. (ed.), *Neural and Intelligent Systems Integration*, John Wiley and Sons, New York, NY. pp. 341–381. [R]

Sun, G.Z., Chen, H.H., Giles, C.L., Lee, Y.C., and Chen, D. (1990). Connectionist pushdown automata that learn context-free grammars. In *Proceedings of International Joint Conference on Neural Networks*, Vol.1, pp. 577–580, M. Caudill (ed.), Lawrence Erlbaum Associates, Hillsdale, New Jersey. [L]

Sun, G.Z., Chen, H.H., Lee, Y.C., and Giles, C.L. (1990). Recurrent neural networks, hidden Markov models and stochastic grammars. in *Proceedings of International Joint Conference on Neural Networks*, Vol.1, pp. 729–734, San Diego, CA. [L]

Suttner, C. and Ertel, W. (1991). Using back-propagation networks for guiding the search of a theorem prover. *Int. J. of Neural Networks Research and Applications*, 2(1):3–16. [J]

Sutton, R.S. (1985). The learning of world models by connectionist networks. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pp. 54-64. Hillsdale, NJ: Erlbaum. [L]

Takagi, H. and Hayashi, I. (1991). Non-driven fuzzy reasoning. *International Journal of Approximate Reasoning*, 5(3):191-212. [F]

Tan, C.L., Quah, T.S., and Teh, H.H. (1991). A neural logic based expert system. In *Proceedings of the Expert Systems Applications Conference*, pp. 301-306. [E]

Taraban, R.M. and Palacios, J.M. (1993). Exempler models and weighted cue models in category learning. In Nakamura, G., Taraban, R., and Medin, D. (eds.), *The Psychology of Learning and Motivation: Categorization by Humans and Machines*, Vol.29, pp. 91-127. Academic Press, San Diego, CA. [S]

Thagard, P. (1989). Explanatory coherence. *Behavioral and Brain Sciences*, 12(3):435-502. [O]

Thagard, P., Holyoak, K.J., Nelson, G., and Gochfeld, D. (in press). Analog retrieval by constraint satisfaction. *Artificial Intelligence.* Also available as: CSL Report 41. Princeton, NJ: Princeton University, Cognitive Science Laboratory. [A]

Tirri, H. (1991). Implementing expert system rule conditions by neural networks. *New Generation Computing*. 10:55-71. [E]

Todd, P.M. and Rumelhart, D.E. (in press). Feature abstraction from similarity ratings: A connectionist approach. In Chauvin, Y. and Rumelhart, D.E. (eds.), *Backpropagation: Theory, Architectures, and Applications.* Hillsdale, NJ: Erlbaum Associates. [L]

Torras, C. (1992). Symbolic planning versus neural control in robots. In Rudomín, P., Arbib, M.A., and Cervantes-Pérez, P. (eds.), *Natural and Artificial Intelligence: A Meeting Between Neuroscience and Artificial Intelligence*, Research Notes in Neural Computing, Vol. 4. Springer-Verlag: Berlin Heidelberg New-York. [J]

Touretzky, D. S. (1989). Chunking in a connectionist network. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 1-8. Hillsdale, NJ: Erlbaum. Also Available as: Technical Report CMU-CS-89-

158. Pittsburgh, PA: Carnegie Mellon University, Computer Science Dept. [R]

Touretzky, D.S. (1990). BoltzCONS: Dynamic symbol structures in a connectionist network. *Artificial Intelligence 46*, 1(2):5–46. Also In Hinton, G. E. (Ed.), *Connectionist Symbol Processing*, Bradford Book, pp. 5–46. MIT Press, Cambridge, MA. [S]

Touretzky, D.S., and Hinton, G.E. (1988). A distributed connectionist production system. *Cognitive Science*, 12(3):423–466. [R]

Touretzky, D. S. and Geva, S. (1987). A distributed connectionist representation for concept structures. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. [S]

Touretzky, D.S. and Hinton, G.E. (1985). Symbols among the Neurons: Details of a Connectionist Inference Architecture. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pp. 239–243. [R]

Touzet, C. and Giambiasi, N. (1991). Application of connectionist models to fuzzy inference systems. In *Parallelization in Inference Systems*, pp. 303–317. Springer Verlag. [F]

Towell, G. and Shavlik, J.W. (1992). Using symbolic learning to improve knowledge-based neural networks. In *AAAI'92*, pp. 177–182. [L, R]

Towell, G. G., Shavlik, J. W., and Noordewier, M. O. (1990). Refinement of approximate domain theories by knowledge-based neural networks. In *AAAI-90*, pp. 861–866. Morgan Kaufmann. [L, R]

Towell, G.G. and Shavlik, J.W. (1993). The extraction of refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101. [L, R]

Tresp, V., Hollatz, L. and Ahmad, S. (1993). Network structuring and training using rule-based knowledge. In Giles, C.L, Hanson, S.J., and Cowan, J.D. (eds.), *Advance in Neural Information Processing Systems 5*. San Mateo, CA: Morgan Kaufmann Publishers. [L, R]

Ultsch, A., Hannuschka, R., Hartmann, U., and Weber, V. (1990). Learning of control knowledge for symbolic proofs with backpropagation networks. In Eckmiller, R., Hartmann, G., and Hauske, G., (Eds), *Parallel Processing in Neural Systems and Computers*, pp. 499–502. Elsevier. [L, R] van Gelder, T. (1989). Compositionality and the explanation of cognitive processes. In *Proceedings of the Annual Conference of the Cognitive Science Society,* pp. 34–41. [O]

van Gelder, T. (1990). Compositionality. *Cognitive Science.* 14:355–384. [O]

Wald, J., Farach, M., Tagamets, M., and Reggia, J.A. (1989). Generating plausible diagnostic hypotheses with self-processing causal networks. *Journal of Experimental and Theoretical Artificial Intelligence,* 2:91–112. [E]

Waltz, D.L. and Feldman, J.A. (eds.) (1988). *Connectionist Models and Their implications,* Ablex. Norwood, NJ. [C]

Waltz, D.L. and Pollack, J.B. (1985). Massively parallel parsing: A strongly interactive model of natural language interpretation. *Cognitive Science,* 9:52–74. [N]

Ward, N. (1993). *A Connectionist Language Generator.* Norwood, NJ: Ablex. [N]

Werbos, P. J. (1992). Neurocontrol and fuzzy logic: Connections and designs. *International Journal of Approximate Reasoning,* 6(2):185–220. [F]

Wermter, S. (1989). Integration of semantic and syntactic constraints for structural noun phrase disambiguation. In *Proc. of Eleventh International Joint Conference on Artificial Intelligence,* Detroit, USA. [N, J]

Wermter S. and Lehnert W.G. (1992). Noun phrase analysis with connectionist networks. In Reilly R. and Sharkey N. (eds.), *Connectionist Approaches to Language Processing,* Hilsdale, NJ: Erlbaum. [N, J]

Whitson, G., Wu, C., and Taylor, P. (1990). Using an artificial neural system to determine the knowledge base of an expert system. In Levine, D. and Aparicio, M. (Eds.), *Neural Networks for Knowledge Representation and Inference.* [E]

Yamaguchi, T. and Imasaki, N. and Haruki, K.. (1990). Fuzzy rule realization on associative memory system. *IJCNN90-2,* p. 720–723. [F]

Yang, Q. and Bhargava, V. (1990). Building expert systems by a modified perception network with rule-transfer algorithms. In *Proc.of International Joint Conference on Neural Networks, Vol.2,* pp. 77–82. [L, E]

Author Index

Subject Index

About the Editors

Ron Sun is currently an assistant professor of computer science at the University of Alabama. He received his Ph.D. in computer science from Brandeis University in 1991. Dr. Sun's research interest centers around the studies of intellegence and cognition, especially in the areas of reasoning, learning and connectionist models. He is the author of 40+ papers, and has written, edited or contributed to 8 books, including the recent monograph: *Integrating Rules and Connectionism for Robust Commonsense Reasoning*, published by John Wiley and Sons. He chaired the symposium on rationality at the 1991 Annual Conference of Society for Psychology and Philosophy. He organized and chaired the AAAI Workshop on Integrating Neural and Symbolic Processes, 1992. He has also been on the program committees of many national and international conferences, such as National Conference on Artificial Intelligence (AAAI-93), International Two-Stream Conference on Expert Systems and Neural Networks, International Symposium for Integrating Knowledge and Neural Heuristics, and has been an invited/plenary speaker for some of them. He was the guest editor of the special issue of *Connection Science* on architectures for integrating neural and symbolic processes. For his paper on integerating rule-based reasoning and connectionist models, he received the 1991 David Marr Award from Cognitive Science Society.

Lawrence A. Bookman is a post-doctoral research scientist at Sun Microsystems Laboratories in Chelmsford, MA. He received his Ph.D. in computer science from Brandeis University in 1992. His research centers on computational models for understanding text, and connectionist and neural models of cognition. He is the author of *Trajectories through Knowledge Space: A Dynamic Framework for Machine Comprehension* published in 1994 by Kluwer Academic Publishers. He was the guest editor of the special issue of *Connection Science* devoted to architectures that integrate neural and symbolic processes, and was co-organizer and co-chair of the AAAI-92 Workshop on *Integrating Neural and Symbolic Processes: The Cognitive Dimension.*