Connectionism and the Problem of Systematicity

Steven Andrew Phillips B.Sc., B.A.(Hons.)

A thesis submitted for the degree of Doctor of Philosophy

Department of Computer Science
The University of Queensland

January 25, 1995

Statement of originality

The work presented in this thesis is, to the best of my knowledge, original, except as acknowledged in the text, and the material has not been submitted, either in whole or in part, for a degree at this or any other university.

Steven Phillips

January 25, 1995

Abstract

Systematicity is a pervasive property of cognitive behaviour (e.g., language and reasoning) whereby the ability to represent (systematicity of representation) and infer from (systematicity of inference) some instances of a structured object extends to other instances conforming to the same structure (Fodor & Pylyshyn, 1988; Fodor & McLaughlin, 1990). The problem of systematicity for a Connectionist approach to cognition is: (1) *Can Connectionist models exhibit systematicity without implementing a Classical cognitive architecture (i.e., without resorting to symbolic representations and processes)?*; and (2) *Can Connectionist models exhibit the necessary acquisition of systematic behaviour (i.e., above chance level)?*

In regard to the first question, after a review of the Classicist/Connectionist debate over alternative explanations for systematicity, I conclude that Connectionist models cannot exhibit systematicity without resorting to some form of Classical compositionality (i.e., tokening of component representations wherever complex representations are tokened). Essentially, Smolensky's (1991) weak (microfeatures) and strong (tensors) compositionality, and van Gelder's (1990) functional compositionality either: (1) cannot support systematicity because component representations are not (uniquely) accessible by other processes; or, (2) can support systematicity, but only by tokening component representations relative to the processes that access them (i.e., by implementing some form of Classical compositionality).

However, I also argue that Connectionism *potentially* offers an explanation for the acquisition of systematic behaviour, which is an issue not addressed in the Classical paradigm since systematicity is built into a Classical architecture. Consequently, the primary concern of this thesis is the second question, which I address by considering two criteria for the necessary acquisition of systematic behaviour, and then evaluating Connectionist models with respect to these criteria on several learning tasks. Thus, systematicity is treated as a problem of generalization rather than representation.

The main results of this thesis concern Hadley's (1993) *strong systematicity* criterion (i.e., generalization to novel component positions) on inference tasks where a network must learn to infer, on request, the components of binary and ternary relations. Through an analysis of internal representations and network learning dynamics I show that, in general, three-layer first-order networks (including the feedforward network, Elman's, 1990 *simple recurrent network*, and Pollack's, 1990 *recursive auto-associative memory*) cannot exhibit strong systematicity on the binary relations inference task. I attribute the lack of strong systematicity to an independence between weights that implement component mappings in each of their possible positions.

In response to the lack of strong systematicity with existing networks, I then developed the *tensor-recurrent network*, which has a dependency between such weights, by incorporating the representational capacity of Smolensky's (1987b) *tensor network*, with the learning capacity of the *simple recurrent network*. Constructing and manipulating tensor representations of complex objects, through the inner and outer product operators, assumes appropriate component, role and cue vectors (for representing and extracting component objects and their roles within a complex object). In the *tensor-recurrent network*, these vectors are learnt by backpropagating an error signal along weighted connections and units implementing the inner and outer product operators. I show that this architecture exhibits the necessary acquisition of systematic behaviour, as defined by Hadley's strong systematicity criterion, on a ternary relations inference task.

In summary, I conclude that:

1. Connectionist models *cannot* demonstrate systematicity without implementing some form of symbolic representation and process.

2. Connectionism *can* provide models, as exemplified by the tensor-recurrent network, with architectural properties that are sufficient for the necessary

acquisition of systematic behaviour from, in part, non-symbolic processes.

The separation of internal representations into component and role vector spaces in the tensor-recurrent network is analogous to a type/token distinction characteristic of symbolic computation. Thus, in terms of systematic behaviour, Connectionism offers a "neural-like" implementation of symbolic processing. However, where Connectionism goes beyond Classicism is in an explanation for the acquisition of systematic behaviour, which is an issue not addressed in the Classical paradigm since systematicity is built into a Classical architecture, not acquired.

List of publications

1. Phillips, S. (1994a). Strong Systematicity within Connectionism: The Tensor-Recurrent Network. To appear in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, Atlanta, GA.

2. Phillips, S. (1994b). Connectionism and Systematicity. In A.C. Tsoi & T. Downs (Eds.), *Proceedings of the Fifth Australian Conference on Neural Networks*, pp. 53-55 Brisbane, Australia.

3. Phillips, S. (1994c). Understanding as generalization not just representation. In J. Wiles, C. Latimer & C. Stevens (Eds.), *Collected Papers from a Symposium on Connectionist Models and Psychology*, pp. 110-111. Technical Report No. 289, Department of Computer Science, The University of Queensland, Australia. A comment on Halford & Wilson's paper: "How far do neural network models account for human reasoning?".

4. Phillips, S. and Wiles, J., (1993). Exponential Generalizations from a Polynomial Number of Examples in a Combinatorial Domain. In *Proceedings of the International Conference on Neural Networks*, pp. 505-508 Nagoya, Japan.

5. Phillips, S., (1993). The Effect of Representation on Error Surface. In P. Leong & M. Jabri (Eds.), *Proceedings of the Fourth Australian Conference on Neural Networks*, pp. 86-89. Melbourne, Australia: Sydney University Electrical Engineering.

6. Phillips, S., (1992). Making a Simple Recurrent Network a Self-Oscillator by Incremental Training. In P. Leong and M. Jabri (Eds.), *Proceedings of the*

Third Australian Conference on Neural Networks, pp. 244-247. Canberra, Australia: Sydney University Electrical Engineering.

7. Phillips, S., Wiles, J. and Schwartz, S., (1991). A Comparison of Three Classification Algorithms on the Diagnosis of Abdominal Pains. In M. Jabri (Ed.), *Proceedings of the Second Australian Conference on Neural Networks*, pp. 283-287. Sydney, Australia. Sydney University Electrical Engineering.

8. Bakker, P., Phillips, S. and Wiles, J. (in press). The 1000-2-1000 encoder: A matter of Representation. *Neural Network World*, 4(5), 527-534.

9. Schwartz, S., Wiles, J. and Phillips, S., (in press). Connectionist, rule-based and Bayesian decision aids: an empirical comparison. In P. Slezak, T. Caelli & R. Clark (Eds.), *Perspectives on Cognitive Science: Theories, Methods and Foundations*, pp. 167-180. Norwood, NJ: Ablex.

10. Schwartz, S., Wiles, J., Gough, I. and Phillips, S., (1993). Connectionist, rule-based and Bayesian decision aids: an empirical comparison. In D. J. Hand (Ed.) *Artificial Intelligence Frontiers in Statistics* (pp. 264-277). London: Chapman & Hall. (NB. An earlier version of Schwartz et al 1994).

11. Bakker, P., Phillips, S. and Wiles, J., (1993). The N-2-N encoder: A matter of Representation. In S. Gielen & B. Kappen (Eds.), *Proceedings of the International Conference on Neural Networks*, pp. 554-557. London: Springer-Verlag.

12. Dennis, S. and Phillips, S., (1991). Analysis Tools for Neural Networks. Tech. Report No. 207, Department of Computer Science, The University of Queensland, Australia

Contents

List of Figures

List of Tables

Preface

Relationship to previous work

Some of the work presented in this thesis has appeared, in part, in a number of published papers. The work presented in Phillips and Wiles (1993) forms the basis of chapter 3. The arguments, results and analysis presented in chapter 4 is a significant expansion of the work that appeared in Phillips (1994b), and chapter 5 provides further analysis and a more detailed explanation of the network architecture that appeared in Phillips (1994a). Also, the work presented in Phillips (1994c) appears in Appendix A.

Two of the techniques used in this thesis have also appeared in previous work. The block encoding technique, for faster training of auto-associators (chapter 3), appeared in Bakker, Phillips and Wiles (1993, 1994), and an explanation of its improvement in training time was based on Phillips (1993). The network analysis tools used in chapter 4 appeared in Dennis and Phillips (1991).

Lastly, there are several works which helped provide an understanding of some of the learning properties of feedforward and recurrent networks. They include, the work on generalization with feedforward networks in Phillips, Wiles and Schwartz (1991), Schwartz, Wiles, Gough and Phillips (1993), and Schwartz, Wiles and Phillips (1994); and the learning capabilities on the simple recurrent network, in Phillips (1992).

Acknowledgements

Throughout the years I have been supported by many people who have helped either directly, or indirectly to produce this thesis. In particular, I thank my supervisor Janet Wiles for all her support prior to and throughout my candidature; for providing valuable discussion and criticism of the work in, and relating to this thesis, and for her constant and critical attention to the writing process.

I would also like to thank my collegues: Simon Dennis, for many interesting and critical discussions on this work; and Paul Bakker, not only for his help and comments on this work, but also in regard to frequent flyer miles.

During my candidature, I have had many useful discussions with researchers who have shared their time and energy to comment and discuss some of the issues relating to this thesis. For their help and support, I would like to thank: Glenda Andrews, Peter Bartlett, Anthony Bloesch, Kerry Chalmers, Bob Colomb, Graeme Halford, Mike Humphreys, Danny Latimer, Ray Lister, David Lovell, Hanna Majewski, Michael Norris, Helen Purchase, Gordon Rose, Guy Smith, Kate Stevens, Julie Stewart and Steven Young.

I would also like to thank Jeff Elman and his research group for providing the *Tlearn* simulator, which was used for the simulations in chapters 3 and 4; and Geoffrey Hinton and his research group for providing and maintaining the *Xerion* simulator, which was used for the simulations in chapter 5.

I am also extremely grateful for the financial support provided by the University of Queensland and the Department of Computer Science. This work was supported by a University of Queensland Postgraduate Research Scholarship, and a Department of Computer Science Postgraduate Research Scholarship. I am also extremely grateful for the excellent facilaties provided by the Department of Computer Science in the form of equipment and support staff, and for financial assistance that has allowed me to attend a number of national and international conferences.

Of course, much help and support was given before I even started this work. I would like to thank my parents for all their support through the years.

Finally, I thank Michiko and Mizuki, for their undying patience and love.

Chapter 1

Introduction

In 1986, a small group of researchers put together two volumes on parallel distributed processing models of cognition (Rumelhart, McClelland, & the PDP research group, 1986; McClelland, Rumelhart, & the PDP research group, 1986) which, as it turned out, repopularized a movement in the study of human and machine intelligence called *Connectionism.*

Connectionist models are typically characterized as collections of interconnected processing units, each specified by some local level process or function (e.g., sigmoid, see Rumelhart, Hinton, & Williams, 1986), which together exhibit some global level behaviour (e.g., hand-written character recognition LeCun, Boser, & Denker, 1989). The analogy, of course, is taken from the human brain, which is similarly organized, hence the term *Artificial Neural Networks* [1].

Since its resurgence in popularity many advocates have proclaimed Connectionism as the new paradigm for cognitive modeling, promising not only models of cognitive behaviour, but also models of cognitive development. For example, Rumelhart and McClelland (1986) have presented a Connectionist network that not only performs the past-tense mapping of English verbs, but also learns this

[1]The term Connectionism as used here refers to neural network research activities directed towards cognitive behaviour (e.g., perception, language, analogy), rather than the more widely used term **Artificial Neural Networks** (or simply, **Neural Networks**), referring to research activities directed at a wider range of engineering applications (including, for example, plant control, stock market prediction, chemical analysis, automatic piloting).

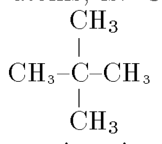
behaviour in a qualitatively similar manner as children[2].

Of central importance to a paradigm or framework for cognition is the ability to provide models that demonstrate such properties as *systematicity*. Briefly, systematicity is the property of human cognition whereby the ability to represent and process structured objects (e.g., *John loves Mary*) is accompanied by an ability to represent and process other objects conforming to the same structure[3] (e.g., *Mary loves John*). That is, cognitive capacities are grouped on the basis of the structural similarity of the objects on which they operate.

It is generally accepted that human cognition is systematic, at least to some significant degree. In general, one does not, for example, observe people capable of inferring that *John went to the store* given that *John and Mary went to the store*, yet unable to infer the structurally similar case of *Mary went to the store* given that *Mary and John went to the store*. What has not been accepted is the implication that the property of systematicity holds for cognitive architecture (i.e., the set of basic data structures and processes employed in cognitive behaviour).

Fodor and Pylyshyn (1988) have argued that systematicity necessitates structured representations (symbol structures) and processes that are sensitive to the structure of those representations. In short, the so-called *Classical* cognitive architecture is a symbol system. A characteristic property of cognition that is posited in a Classical cognitive architecture is that the tokening (inscription) of representations of compositional objects necessarily entails the tokening of representations of all component objects.

[2]Although, the accuracy of this work as a model of the acquisition of past-tense has been strongly criticized (Pinker & Prince, 1988; Marcus, Brinkmann, Clahsen, Wiese, Woest, & Pinker, 1993).

[3]The term structure is taken to mean some invariant property of the relationship between components of complex objects. For example, in chemistry, two types of molecules *n*-Pentane and Neopentane are constructed from the same type and number of atoms (i.e., five carbon atoms and twelve hydrogen atoms), yet they differ structurally due to the difference in the spatial relationship between their component atoms. The structure of an *n*-Pentane molecule, the spatial relationship between its components atoms, is: CH_3–CH_2–CH_2–CH_2–CH_3. In contrast, the structure of a Neopentane molecule is:  (Brown & LeMay, 1981). In the context of representations, structure is regarded as some invariant property of the relationship representational components.

Smolensky (1987b, 1987a, 1990, 1991) and van Gelder (1990) have argued for a more general Connectionist notion of compositionality[4] based on distributed (vector) representations and processes. In a Connectionist cognitive architecture, representations of compositional objects are vectors distributed over a number of processing units. Connectionist representations are non-Classical in that the representation of a compositional object does not necessarily entail representations of the object's components. Thus, in this respect Connectionism, it is claimed, is an alternative to the Classical conception of cognitive architecture. For example, Elman's (1990) simple recurrent network is capable of constructing a single vector representation of a sentence where the representations of individual words do not "appear"[5] in the representation of the sentence in which they are contained. Yet, the network is able to operate, often termed *wholistically*, on the vector representation of the sentence to extract word-level information such as plurality.

Fodor and McLaughlin (1990) have rejected the so-called Connectionist alternative concept of compositionality on the basis that either: (1) vector representations are simply points in space with no internal structure, and therefore cannot account for systematicity since component representations of complex objects are not accessible to other processes; or (2) component representations are accessible by tokening (i.e., inscribing) a vector representation of each component object whenever the complex representation is inscribed, in which case, the Connectionist architecture implements a Classical architecture. Thus, Fodor and McLaughlin concluded, Connectionism is at best an implementation framework for Classical cognitive architectures, and consequently, has nothing to offer in the way of alternative theories of cognition.

This conclusion is, however, far from unanimously accepted. Furthermore, it is not clear as to the grounds on which Fodor and McLaughlin rejected the Connectionist proposals. On the one hand they accept the *possibility* of Connectionist

[4]That is, the method by which a representation of a compositional object is constructed from the representations of its components.

[5]In the sense that no portion of the sentence vector equals any vector representation of its components. For example, the vector $\vec{A} = (3\ 2)$ does not appear in the vector $\vec{B} = (9\ 1\ 7)$.

architectures exhibiting systematicity without resorting to Classical compositionality (p. 202), yet on the other hand they reject such possibilities on the grounds that it is not sufficient to show how systematicity is *possible* given the assumptions of a (Connectionist) architecture, one must also show how systematicity is *necessary* given those assumptions (p. 202). The confusion arises because they do not specify a criterion by which a model is said to necessarily exhibit systematicity.

Given the lack of consensus over the conclusion that Connectionism is at best an implementation framework, an appropriate question to ask is:

- *Can Connectionism provide non-Classical models (i.e., ones that do not rely on symbolic representations and processes) that exhibit systematicity?*

After reviewing, in detail, the Connectionist/Classicist debate with respect to this question, in **chapter 2**, it is concluded that: *Connectionist architectures cannot exhibit systematicity without implementing Classical compositionality.*

However, it is also argued that the limitation of Fodor and Pylyshyn's thesis, and subsequently, Fodor and McLaughlin's rebuttal is that their criterion for systematicity is grounded solely in terms of computational capacity (i.e., in terms of the functions computable by an architecture). Consequently, systematicity is realizable by many computationally sufficient architectures, including the possible Connectionist alternatives. Connectionism, however, is also concerned with the modeling of cognitive development. Thus, it is argued, a "potential" contribution of Connectionism to cognitive theory is in an explanation for the necessary acquisition of systematic behaviour. Therefore, the problem that systematicity poses for Connectionism, and the question of primary concern in this thesis is:

- *Can Connectionism provide models that exhibit the necessary acquisition of systematic behaviour?*

The approach taken in an attempt to address this question is to: first, specify a suitable criterion by which a model is said to necessarily acquire systematic behaviour; and second, evaluate Connectionist models with respect to this criterion on learning tasks designed to test their capacity to acquire systematic behaviour.

The first criterion, taken from computational learning theory (Valiant, 1984), is based on the amount of computational resource required to learn a systematic behaviour to a high degree of accuracy. In **chapter 3**, a Connectionist architecture called the feedforward network is shown, by analysis and simulation, to meet this criterion on a task designed to test systematicity.

However, it is then suggested that this criterion may be too weak in that the amount of resource required is more than that required by people. Consequently, a second criterion is used, called strong systematicity (Hadley, 1993), which is based on linguistic evidence of generalization over structurally similar sentences. In **chapter 4**, it is shown, by analysis, that the same network cannot exhibit the acquisition of systematicity as defined by this criterion on the same task. Strong systematicity is then evaluated on a number of other models based on a recurrent network architecture. Briefly, none of the models examined demonstrate strong systematicity with respect to an inference task whereby the model is required to infer components from complex objects.

The lack of strong systematicity in the models examined prompted the design of a further architecture, called the tensor-recurrent network, specifically to address the issue of the necessary acquisition of systematicity. In **chapter 5**, the tensor-recurrent network is presented and shown, by simulation, to exhibit strong systematicity with respect to the inference task.

A discussion of these results and their relationship to the Classical paradigm is given in **chapter 6**. Finally, conclusions are drawn, and further work is suggested in **chapter 7**.

1.1 Statement of thesis

Having reviewed the Classicist and Connectionist arguments relating to systematicity, and having tested and analyzed a number of Connectionist architectures for their capacity to acquire systematic behaviour, the two main conclusions of this thesis are:

1. Connectionist models cannot demonstrate systematicity without implementing some form of symbolic representations and processes.

2. Connectionism can provide models, as exemplified by the tensor-recurrent network, with architectural properties that are sufficient for the necessary acquisition of systematic behaviour from, in part, non-symbolic processes.

Chapter 2

Systematicity: What is the problem?

2.1 Introduction

As mentioned in chapter 1, Fodor and Pylyshyn have used the systematicity property to argue against the Connectionist approach to cognitive modeling. Yet, as also pointed out, it is not clear that systematicity cannot be addressed within the Connectionist framework. The purpose of this chapter is to establish the central problem that systematicity poses for the Connectionist approach to cognition; and subsequently, the question that this thesis attempts to address. Therefore, an appropriate place to begin is with a review of the Classicist/Connectionist debate over the implication of systematicity to cognitive modeling with the following question in mind: *Can Connectionism provide non-Classical models that exhibit systematicity?*

In section 2, the Classical explanation for systematicity is presented. The Classical view is that to exhibit systematicity a cognitive architecture must have two properties: (1) structured representations (symbol structures); and (2) processes that are sensitive to the structure of those representations. The claim that Connectionism provides an alternative view of cognition turns at one point on an alternative explanation for systematicity; one that does not rely on the Classical notion of compositionality (i.e., the tokening or inscription of component representations whenever complex representations are tokened). In section 3, I argue that Connectionist models cannot exhibit systematicity without implementing Classical compositionality. However, I also argue that the direction along which Connectionism "potentially" extends the Classical theory is in the acquisition of systematic behaviour. The question then arises: *Can Connectionism provide models that exhibit the necessary acquisition (learning) of systematic behaviour?* This question, which is the central concern of this thesis, and the method used to address it are given in section 4. Finally, a summary is provided in section 5.

2.2 Systematicity and its Classical explanation

Systematicity is a property of cognitive beings in that their capacity for certain cognitive behaviours is grouped on the basis of the structural similarity of these behaviours (Fodor & Pylyshyn, 1988). Fodor and Pylyshyn distinguish two aspects of systematicity: *systematicity of representation* and *systematicity of inference*.

Systematicity of representation is the property that, if one can represent a structured object (e.g., *John loves Mary*[1]), then one also has the ability to represent the structurally similar object *Mary loves John*.

Similarly, systematicity of inference is the property that, if one can infer, for example, that *Tom went to the store* from the sentence *Sue and Tom went to the store*, then one also has the ability to infer that *Sue went to the store* from the sentence *Tom and Sue went to the store*.

The ability to make such generalizations is based on a similarity of structure between compositional objects (i.e., objects that are composed of other simpler objects). Thus, systematicity is a property at the level of compositional objects.

For example, in the domain of scene description, it is reasonable to expect that people are able to verbalize the word "cat" on seeing the object *cat*, but without further experience are unable to verbalize the word "platypus" on seeing

[1]The *italicized* word is used to refer to the object, not its representation.

the object *platypus*. Because, in this case, the relationship between one object and its label (name) is independent of the relationship between the other object and its associated label. In other words, the two objects are atomic (no further structure) with respect to their labels[2]. However, what one does not find, in general, are people capable of verbalizing the sentence "the cat chases the dog" on seeing the compositional object *cat chases dog*, but unable to verbalize the sentence "the dog chases the cat" on seeing the compositional object *dog chases cat*. Because, in the compositional case, there is a relationship between the two objects and their corresponding labels (i.e., the structure *agent-action-patient*). This point has an important consequence when deciding on appropriate external representations for atomic objects and will be returned to when examining some Connectionist attempts to demonstrate systematicity.

At this point it is important to identify one misconception regarding systematicity. I use an example[3] from the reading-aloud domain to make my point. In English, the "w" component of the word "sweet" is vocalized, however, in the word "sword" the "w" component is silent. This inconsistency is *not* a case for the lack of systematicity of human cognition. It *is* a case for the lack of systematicity in one human cognitive domain (namely, correct pronunciation of English words). The fact that people can, upon request, produce the "w" sound in the word "sword" (i.e., by producing the "s" sound followed by the "word" sound) *is* an example of the systematicity of human cognitive capacities. Another example concerns the interchangeability of nouns in the agent and patient positions. For example, people capable of saying the sentence "John ate the cheese" are unlikely to be heard saying, "the cheese ate John". Again, the lack of apparent systematicity in this domain does not preclude people from understanding the sentence in the sense that they could answer questions like: "What did the cheese eat?"; or "Who ate John?".

Although the extent to which cognition is systematic is arguable, at least in the

[2]Note that the term atomic is a relative one. The same objects would be considered structured when labeled as "quadrapeds", in which case one may expect generalization from one object to the other based on their *limb* component objects.

[3]Presented in a seminar on word recognition by Sally Andrews.

linguistic domain, there is ample evidence of systematicity in adults (see Hadley, 1993, for a review) and for that matter perhaps even children (Pinker, 1984).

That systematicity is a significant property of cognitive behaviour is, in general, not disputed by Connectionists. What is contended, is the implication of systematicity in terms of a cognitive architecture (i.e., the basic algorithms and data structures of a cognitive system utilized in the performance of cognitive tasks). Or in other words, it is agreed that cognitive behaviour is significantly systematic at the computational level, but disputed as to what systematicity implies at the algorithmic level[4].

The distinction between the computational and algorithmic levels is illustrated in the following example. Consider a function called *minimum* that returns the smallest number from a set of numbers. The function can be described at the computational level as: $minimum(S) = e, \text{where } e \leq e_i \ \forall e_i \in S$. The function *minimum* takes a set of numbers S and returns the smallest number e from the set.

At the algorithmic level, an algorithm to compute the function could be described as: removing an element (e) from the set (S) leaving a new set (S') and returning, either the element (e) if the new set (S') is empty; or the smaller of e and the *minimum* of S'.

The value of a computational level description or specification of some behaviour is that it constrains the possible algorithms that can implement such behaviour. The value of systematicity as a computational level property is that it constrains the class of possible cognitive architectures to those that exhibit this property. For example, a cognitive architecture should be capable of exhibiting the behaviour illustrated in Figure 2.1. The contention is over what characterizes architectures that exhibit such systematic behaviour. The predominant view in cognitive science, what Fodor and Pylyshyn (1988) and Fodor and McLaughlin (1990) term *Classicism*, is that to support such properties as systematicity a cognitive architecture

[4]Referring to Marr's (1982) levels of analysis where the computational level is concerned with *what* is being computed, and the algorithmic level is concerned with *how* it is being computed. There is also an implementation level concerned with how the algorithmic level is physically realized.

must make use of:

1. a combinatorial syntax (or structured representations); and
2. processes that are sensitive to the structure of those representations (Fodor & Pylyshyn, 1988; Fodor & McLaughlin, 1990).

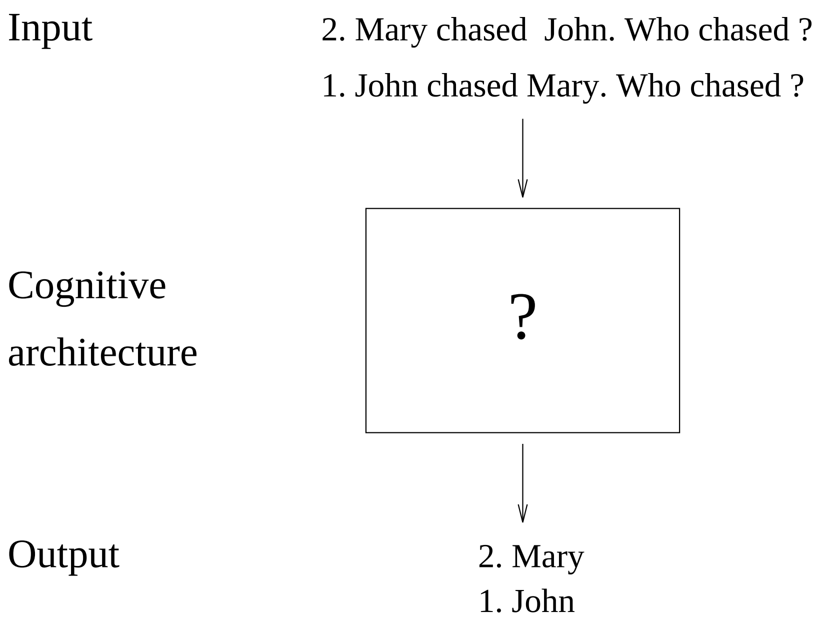

Input
2. Mary chased John. Who chased ?
1. John chased Mary. Who chased ?
Cognitive architecture
?
Output
2. Mary
1. John

Figure 2.1: Systematicity: a computational level property.

A combinatorial syntax is a symbol-level description (i.e., independent of the physical realization of the symbol) of the relationship between representations of compositional objects (called complex representations) and the representations of their component objects (called component representations, or constituents[5]). For example, in the list of syntactic relations:

S $\rightarrow$ N Vt N

N $\rightarrow$ John

N $\rightarrow$ Mary

Vt $\rightarrow$ chases,

[5]The term constituents was used in Fodor and Pylyshyn (1988) and van Gelder (1990).

"S" denotes the complex representation of a compositional object consisting of three components: an action, an agent (performer of the action), and a patient (recipient of the action); "N" denotes the constituent representation of the agent and the patient; and "Vt" denotes the constituent representation of the action. The symbol "N" has as a possible component the symbol "John" or "Mary", and the symbol "Vt" has as a component the symbol "chases".

In this example world, all four compositional objects can be represented by making all four possible expansions as defined by the syntax rules. Thus, systematicity of representation is captured by the combinatorial nature of the rules of syntax. Since each symbol is replaceable by any one of its possible constituent symbols, all possible representations are realizable.

The need for a combinatorial syntax constrains the way in which a cognitive system puts these representations together. It says that the processes for constructing representations are such that: firstly, there is a distinction between structurally atomic (e.g., "N", "John") and structurally molecular (e.g., "S") representations; secondly, a representation of a compositional objects has, as its parts, representations of the object's components; and thirdly, the relationship between an object's components is captured by the relationship between their corresponding constituents. For example, the agent component is identifiable as it is always the first constituent symbol. The particular symbol order is not important, only that the order is used consistently. By mirroring the structural relations in objects with analogous structural relations in their corresponding representations, a combinatorial semantics is captured by a combinatorial syntax.

In addition to a combinatorial syntax, the Classical view is that a cognitive architecture must also have processes that are sensitive to the way in which these representations are structured. Structure sensitive processes manipulate complex representations on the basis of their constituency relations, not on the basis of particular constituents. In the *John-chases-Mary* example, the agent of the action is always the first constituent symbol of the compositional object's complex representation. Consequently, regardless of whether the agent is *John* or *Mary*, a process that can determine the first constituent symbol from a complex representation can do so for all compositional objects in this domain.

Together, structured representations and processes that are sensitive to the structure of those representations are two constraints on cognitive architecture that explain the systematicity property as exemplified in Figure 2.1.

Before presenting arguments for and against the Connectionist alternatives, it should be pointed out that the Classicist view is not a statement about the physical realization of these representations and processes; it is a statement about the relationships between their physical realizations. The physical structural relationships of the brain correspond to symbolic structural relationships of the mind. For example, the symbols John, Mary, and chases may be physically realized as **John**, **Mary**, and **chases** respectively; or alternatively, **1 0 1 1**, **0 1 1 0**, and **0 1 0 1** respectively. Additionally, the process for combining these atomic representations may be concatenation. In which case, the complex symbol "John chases Mary" may be physically instantiated as **John chases Mary**, or **1 0 1 1 0 1 1 0 0 1 0 1**. As well the associated structure sensitive process for extracting the doer of the action would be one that scans for the first space, in the first instance; and one that counts the first four digits in the second instance.

Structured representations and structure sensitive processes make up what is called a symbol system, and its physical realization is called the physical symbol system (Newell, 1980). It is the symbol system and its distinction from its physical implementation that characterizes Classical cognitivism (Pylyshyn, 1980). The Classicist's approach to understanding cognitive behaviour is determining the nature of these symbol structures and processes.

2.3 Is there a Connectionist alternative?

Connectionism, at least from a cursory glance, appears to offer a radically different explanation of cognitive behaviour: An explanation where cognitive behaviour is seen as an emergent property of the interaction of simple units governed by numeric equations. However, for Connectionism to be a successful approach to cognition it must ultimately explain systematicity. It is because people are systematic that their capacities extend to such a vast (combinatorial) number of examples. A framework that cannot address systematicity is unlikely to scale from small experimental domains to real-world human domains.

The focus of architectural design on local-level interactions between simple processing units is not unique to Connectionism. A similar approach has been used by Brooks (1991a, 1991b), in what is called a *subsumption* architecture, in the design of "intelligent" robots. Like Connectionism, Brooks's approach has also been criticized for its scalability (Kirsh, 1991a). Thus, systematicity is not only important for cognitive modeling, but it is also of practical significance.

To be an interesting approach, Connectionism must offer something that Classicism does not. Consequently, the first concern is whether or not Connectionism can offer an explanation for systematicity that does not rely on structured representations and structure sensitive processes.

2.3.1 The horns of Fodor

Fodor and Pylyshyn's position is that to explain systematicity it is necessary to posit structured representations, and processes that are sensitive to the structure of those representations. Furthermore, they argued that Connectionism fails to provide an alternative *theory* of cognition because either

- connectionist models do not use structured representations and structure sensitive processes, in which case, they cannot demonstrate systematicity (*systematicity horn*); or,

- connectionist models do use structured representations and structure sensitive processes to implement systematicity, in which case, Connectionism is an implementation theory of Classical cognitivism (*implementation horn*) (Fodor & Pylyshyn, 1988; Fodor & McLaughlin, 1990).

At this point it should be emphasized that Fodor and Pylyshyn fully endorse the possibility of Connectionist networks that demonstrate systematicity. And, clearly, given the result that there exists a Connectionist network with a finite number of processing units that implements a universal Turing machine (Siegelman & Sontag, 1991), it must be possible to construct a network that exhibits systematicity. However, this result alone is not a refutation of Fodor and Pylyshyn's thesis (as has been argued by Chalmers (1990b, 1993)), because one must also avoid the implementation horn of their argument. (See, also, Butler, 1993, for a similar comment on Chalmers.)

2.3.2 Weak and strong compositionality

The principal defendant of the Connectionist position in this debate over the import of Connectionism has been Smolensky (1987b, 1987a, 1990, 1991). Smolensky's position is to provide two alternative ways in which Connectionist models make use of compositional representations. They are microfeatures and the tensor product, which Smolensky (1991) labels as *weak* and *strong compositionality*, respectively.

Weak compositionality is the representation of objects as a collection of microfeatures, where each microfeature is a unit that becomes activated only in the presence of some feature (component) of an object. For example, the complex object *John chased Bill* can be represented as the set of active units {John, chased, Bill}. The structurally similar object *Bill chased John* would be represented by the set $\{Bill, chased, John\}$. However, as sets do not preserve order both objects have the same representation. Consequently, it is not possible to uniquely determine, for example, who is the agent.

A system that represents complex objects by representing only the components without regard for their structural relationship is inadequate on the basis that it fails to differentiate objects with the same components but different structural relationships. This example illustrates why Fodor and Pylyshyn talk of two necessary dimensions in a representational space: one to record the presence or absence of a component; the other to record its relationship to other components.

Alternatively, one could augment the class of feature detectors with specific John-agent, John-patient, Bill-agent and Bill-patient units. In this case, the two objects can be represented as {John-agent, chased, Bill-patient} and {Bill-agent, chased, John-patient}, respectively. Now, suppose there is another complex object, *Mary chased Mark* resulting in the set of activated units {Bill-agent, chased, John-patient, Mary-agent, Mark-patient}. At this point it is not possible to determine whether "Bill chased Mark", or "Mary chased Mark"; or whether "Bill chased John", or "Mary chased John". Again, the class of feature detectors could be augmented to include, for example, the Bill-chased-John unit. However, the problem with this scheme, as Fodor and Pylyshyn have correctly pointed out, is that the number of feature-detecting units grows rapidly with the complexity of the object. In fact, considering n-ary relations, the number of units required to uniquely represent all possible relations between k possible objects is: k^n (i.e., the number of units grows exponentially with the arity of the relation).

Clark (1991) suggested that the number of microfeature units could be reduced by utilizing existing microfeatures from the component objects. For example, in the complex object *John hit Bill* an existing microfeature could be "tears"[6] from which the system could deduce that *Bill* was the recipient of the action. However, such features must still be bound to Bill, and so there needs to be available one unit for every possible component object. Furthermore, it is not clear that such information will always be available to differentiate possible constituents, particularly if complex objects are communicated verbally rather than visually. I conclude that such a scheme either does not avoid the explosion of microfeature units, or does not exhibit systematicity.

The alternative to using sets of purely atomic features is to allow individual features to themselves be sets. For example, the object *John chases Bill* could be represented as the set {{John, agent}, chases, {Bill, subject}}, which is distinct from the set {{Bill, agent}, chases, {John, subject}} representing the object *Bill chases John*. However, this scheme is characteristically Classical in that there is a distinction between atomic and molecular representations, and the constituents are tokened wherever the complex representation is tokened. Consequently, as Fodor

[6]Extra visual information arising from the fact that being hit by John caused Bill to cry.

and Pylyshyn pointed out, this scheme does not avoid the implementation horn of the argument.

Weak compositionality, I conclude, fails to avoid the two horns of the argument. Either, all complex objects of a particular structure cannot be (uniquely) represented (systematicity horn); or the representation of all complex objects requires Classical compositionality - the tokening of constituents with the tokening of complex representations (implementation horn).

Strong compositionality is the representation of complex objects by tensors. In Smolensky's case, the tensor representation is constructed as the sum of the outer products of each component-role representation pair. For example, the object *John chases Bill* could be represented by the tensor construction:

$$\vec{T}_{John\ chases\ Bill} = \vec{V}_{John} \otimes \vec{R}_{agent} + \vec{V}_{chases} \otimes \vec{R}_{action} + \vec{V}_{Bill} \otimes \vec{R}_{patient}$$

where $\vec{T}_{John\ chases\ Bill}$ is the tensor representation of the object *John chases Bill*; $\vec{V}_{John}$, $\vec{V}_{chases}$ and $\vec{V}_{Bill}$ are the vector representations of the component objects *John*, *chases* and *Bill*, respectively; and $\vec{R}_{agent}$, $\vec{R}_{action}$ and $\vec{R}_{patient}$ are the vector representations of each component object's respective role.

Alternatively, complex objects can be represented as the running outer product of each component object representation (Halford, Wilson, Guo, Gayler, Wiles, & Stewart, 1994). In this case, the object *John chases Bill* is constructed as:

$$\vec{T}_{John\ chases\ Bill} = \vec{V}_{John} \otimes \vec{V}_{chases} \otimes \vec{V}_{Bill}.$$

Tensor representations were intended by Smolensky as an example of Connectionist representations that are non-Classical, and therefore non-implementational. Tensors "appear" to offer a form of non-Classical compositionality in the sense that constituents do not occur within the complex representation. For example, given the constituent-role pairs $\vec{V}_1 = (3\ 1\ 4)$, $\vec{R}_1 = (-2\ 3)$ and $\vec{V}_2 = (2\ 3\ 1)$, $\vec{R}_2 = (3\ 2)$; the resulting complex representation:

$$\vec{V}_1 \otimes \vec{R}_1 + \vec{V}_2 \otimes \vec{R}_2 = \begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix} \begin{pmatrix} -2 & 3 \end{pmatrix} + \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix} \begin{pmatrix} 3 & 2 \end{pmatrix} \qquad (2.1)$$

$$\tag{2.2}$$

$$= \begin{pmatrix} 0 & 13 \\ 7 & 9 \\ -5 & 14 \end{pmatrix} \tag{2.3}$$

does not contain in any one of its rows or columns a vector that equals any one of the constituent vectors from which it was constructed. However, does this compositionality scheme account for systematicity?

The problem with tensor constructions, as Fodor and McLaughlin (1990) have pointed out, is that there is potentially an infinite number of constituent combinations that result in the same tensor representation, and therefore it is not possible to determine which combination was used to construct the tensor representation. To see the force of this point, suppose a system has represented as a tensor a complex object consisting of two component objects and the system is required to extract one of its constituents for further processing. That is,

$$\vec{a} \otimes \vec{b} = \vec{T}_{a,b}$$

where $\vec{a}$ and $\vec{b}$ are the vector representations of the first and second component objects, respectively. Now, suppose two different objects have vector representations $\vec{a}' = \frac{1}{k}\,\vec{a}$ and $\vec{b}' = k\vec{b}$, where $k \neq 0, 1$. The corresponding representation of the complex object that is composed of these two objects is,

$$\vec{a}' \otimes \vec{b}' = \frac{1}{k} k \vec{a} \otimes \vec{b} = \vec{T}_{a,b}.$$

Since, one complex representation stands for two different complex objects it is not possible to uniquely determine the component objects. Furthermore, in general, there are an infinite number of possible decompositions for the same complex representation. Thus, the value of the vector (tensor), in itself, provides insufficient information to determine which of the possibly infinite components were the actual components. In this case, the tensor does not allow for systematicity, because it does not give unambiguous access to its components. Therefore it fails to avoid the systematicity horn.

The alternative is to provide one of the constituent vectors to a process that performs the inner product of the constituent with the tensor to extract the other constituent vector. However, in doing so, one must maintain the representation of the constituent whenever one is processing the complex representation (i.e., one must token the constituent every time one tokens the complex representation). Therefore, this alternative fails to avoid the implementation horn of the argument.

In both weak and strong compositionality, Smolensky has presented compositionality schemes that are non-Classical in that the representations of complex objects are tokened without having to token representations of their component objects. However, both schemes fail to explain systematicity because the components are no longer uniquely determinable.

2.3.3 The concatenative/functional distinction

The tensor solution is an example of what is generally regarded as structured representations without the necessity for them to be symbol structures. Van Gelder (1990) characterizes this distinction as the difference between concatenative and functional modes of compositionality. That is, modes of combining representations of component objects to form a new representation standing for the corresponding compositional object. Furthermore, he argues, that because Connectionism is committed to a functional compositionality scheme it is an alternative theoretical framework to Classical cognitivism. Before examining this claim the two modes of compositionality are described.

Concatenative compositionality is the spatial or temporal juxtapositioning of representational components to form complex representations. For example, consider the complex object *Tom chased Sue* which is composed of the atomic objects *Tom*, *chased*, and *Sue*. Now, suppose these atomic objects have the representations **Tom**, **chased**, and **Sue**, respectively. Then a corresponding concatenative compositional representation may be **Tom chased Sue**. The point to note is that the constituent (e.g., **Tom**) remains unchanged within the complex representation.

Functional compositionality is a more general form of compositionality whereby the representation of a complex object is the result of some function that takes the constituent representations as its arguments. Consider again the *Tom chased Sue* example. This time suppose the objects *Tom*, *chased*, and *Sue* are represented by the numbers **1**, **3**, and **5**, respectively. A possible functional composition of these vectors is **267** under the function: $f(x, y, z) = 7^0x + 7^1y + 7^2z$, where x, y and z are numeric representations of the three components. Constituents can be uniquely extracted by the functions:

$$\begin{aligned} f_1(R) &= R \bmod 7 &&= x \\ f_2(R) &= (R \text{ div } 7)\ mod\ 7 &&= y \\ f_3(R) &= R \text{ div } 7^2 &&= z \end{aligned}$$

Although functional compositionality contains concatenative compositionality as a special case, and is therefore a more general conception of compositionality, it is tangential to the Classicist/Connectionist debate for two reasons. Firstly, Classicism was never committed to a concatenative style of compositionality.

> "A suitable definition [of a physical instantiation mapping (F)] might contain the statement that for any expressions P and Q, F[P&Q] = B(F[P],F[Q]), where the function B specifies the physical relation that holds between physical states F[P] and F[Q]. Here the property B serves to physically encode, (or 'instantiate') the relation that holds between the expressions P and Q on the one hand, and the expressions P&Q on the other."
>
> (Fodor and Pylyshyn, 1988: p14)

Thus, the function B is exactly what van Gelder calls a functional mode of compositionality taking the component representations F[P] and F[Q] as arguments and returning a representation of the compositional object P&Q. The point is that the Classical picture leaves the function B unspecified and is therefore neutral as to the mode of compositionality used by the brain. Function B is unspecified as it is irrelevant to the cognitive level, and only relevant to the implementation level. The independence of these two levels is foundational to a Computational Theory of Mind (Pylyshyn, 1980). That Classicism is not committed to a concatenative compositionality is re-emphasized in two further places:

> "... the relation that is physically realized is *functional* adjacency, there is no necessity that physical instantiations of adjacent symbols be *spatially* adjacent."
>
> (Fodor and Pylyshyn, 1988: p57; emphasis in the original)

> "Though we shall generally consider examples where complex symbols literally *contain* their Classical constituents, the present condition means to leave it open that symbols may have Classical constituents that are not among their (spatio-temporal) parts. (For example, so far as this condition is concerned, it might be that the Classical constituents of a symbol include the values of a 'fetch' operation that takes the symbol as an argument.)"
>
> (Fodor and McLaughlin, 1990: p186; emphasis in the original)

The contention over Classical constituents, that is, when is a component tokened within the tokening of its whole, is analogous to Kirsh's (1991b) computational definition of explicitness of information. Kirsh defines explicitness of information within a larger context as information that is accessible within a reasonable (constant in Kirsh's case) amount of time. The important point that Kirsh makes is that

> **the accessibility of information is dependent on, or relative to, the processes that access that information**.

The point that has not been previously made by either the Classicists or the Connectionists is that tokening of a constituent, like Kirsh's explicitness of information, is relative to the process extracting that constituent.

In the tensor example from equation 2.3, let an access function f_{R_1} be defined as:

$$f_{R_1}(\vec{T}) = \frac{1}{13}(\vec{T} \odot (-2\ 3)).$$

Then, constituent $\vec{V_1}$ is explicitly tokened relative to $f_{R_1}(\vec{T})$ since

$$f_{R_1}\left(\begin{bmatrix} 0 & 13 \\ 7 & 9 \\ -5 & 14 \end{bmatrix}\right) = \begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix}.$$

The function f_{R_1} specifies a particular projection or view of the complex representation. When the complex representation is viewed along this projection the constituent becomes explicit (i.e., tokened).

The point is that the mode of compositionality is irrelevant beyond there being explicit representations of components within complex representations relative to the processes that operate on those representations.

The second reason why the distinction between concatenative and functional modes of compositionality is irrelevant is that the modes of compositionality, in themselves, do not guarantee systematicity. Or, in other words, a model that makes use of a concatenative compositionality scheme can be just as unsystematic as a model under some other functional compositionality scheme. An example is given to illustrate this point.

Suppose that the objects *John*, *loves* and *Mary* are encoded as binary-valued vectors **1 0 1**, **0 1 0** and **1 0 1 0 1**, respectively. Then, under a concatenative compositionality scheme a possible encoding of the object *John loves Mary* is **1 0 1 0 1 0 1 0 1 0 1**. In order to extract the constituents a process must know at which point to truncate the vector. However, it is not possible to determine with certainty these points without additional information or assumptions as to the way in which the constituents were concatenatively constructed. For example, under the same compositionality scheme, the compositional object *Mary loves John* has the same complex representation (i.e., **1 0 1 0 1 0 1 0 1 0 1**). Additional information could be provided, for example, in the form of separation markers (say a component with value 0.5). A similar problem arises with functional compositionality schemes. The point is that the processes operating on compositional representations must have some *knowledge* of the way in which constituents are brought together in the first place.

It could also be argued that a commitment to concatenative compositionality is simultaneously a commitment to discrete representations and processes; and since the brain is not a discrete device, the mode of compositionality cannot be concatenative, and therefore cannot be symbolic (i.e., Classical). That the discrete/continuous dimension is orthogonal to the issue of whether cognitive architecture is symbolic is illustrated here by what I term as the *beam scattering analogy.*

In a beam of light there are photons with wavelengths that may vary over some continuous (though not necessarily uniform) distribution. When passed through a prism the photons are scattered by an angle that varies monotonically with wavelength. Consequently, if one were to record the scattering with some photoelectric device that measures the incident light, one would observe intensity as a continuous function of angle. Now, suppose that constituents are encoded at points of zero intensity gradient. The magnitude of light intensity at the maximum, minimum and inflection points of the angle-intensity curve encodes the constituent. In this way, constituents may be concatenated as a continuous waveform without the necessity for the constituents to be confined to discrete fixed-width spatial locations (see Figure 2.2). The point is that concatenative compositionality is not limited to, and therefore does not equate to, discrete constituents. So the argument that cognition could not be symbolic because it could not operate over discrete representations does not follow.

The conclusion taken from a re-examination of this debate is that the **implementation horn of the argument leaves no room for an alternative theory**. Either constituents are tokened (relative to their access processes) with their complex representations, in which case one is implementing a Classical ar-

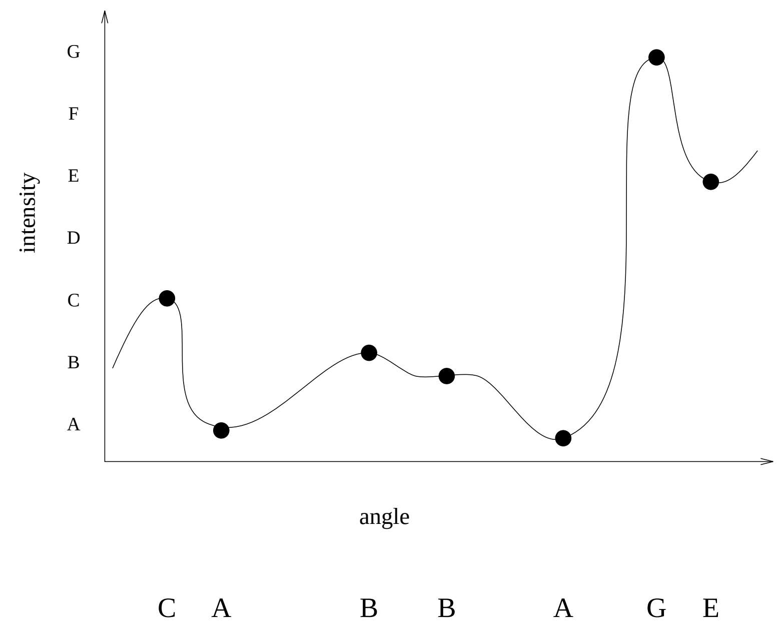


Figure 2.2: Beam scattering analogy: An example of continuous concatenative compositionality where constituents are encoded as intensity magnitudes at the zero-gradient points of an angle-intensity curve.

chitecture; or, constituents are not tokened, in which case one cannot account for systematicity simply because the constituents are not uniquely determinable. So the question becomes, just what (if anything) can Connectionism contribute in terms of cognitive theory?

2.3.4 A "potential" contribution of Connectionism

Classical theory of cognition is based on the assertion that the cognitive level is supported by, but independent of the implementation level, where the cognitive level consists of structured representations (standing for properties of the external world), and processes that operate over these representations. Furthermore, a cognitive state is the result of some process on some internal (structured) representation. From a Classicist's perspective, the proper concerns at the cognitive level of theorizing are determining these processes and representations.

There are, arguably, two ways in which the Classicist picture is (potentially) limited. The first limitation is that it is not at all certain that cognitive behaviour is *only* a consequence of cognitive level representations and processes. Take for instance Fodor's (1975) own example:

> "A man wishes to be reminded, sometime during the day, to send a message to a friend. He therefore puts his watch on upside down, knowing that he will glance at it eventually and that, when he does, he will think to send the message. What we have here is, presumably, a straight forward causal connection between two states ..., but not a kind of connection cognitive psychology has anything to say about. Roughly, although the mental states are causally connected, they aren't connected by virtue of their *content*; compare the case of the man who is reminded to send a message to his friend when he (a) hears and (b) understands an utterance token of the type 'send a message to your friend'."
>
> (Fodor, 1975: p203, emphasis in the original)

A second example[7] is illustrated in the form of the following addition task. Present the sequence of numbers listed in Figure 2.3(a), one at a time, to a subject and request the subject to perform a running additive total as each number is presented. After the last number is presented, ask the subject to respond with the total. Interestingly, some subjects (though not necessarily all) will incorrectly respond with 5000 instead of 4100. Possibly, some perceptual stimulus, perhaps the alternating rhythm of the 1000s, or the predominance of 0s, interacts with the symbolic process of addition. However, the subjects are more likely to respond correctly, given the same numbers but in a different order (Figure 2.3(b)).

A third example is due to van Gelder and Niklasson (1994). They used the results of a study by Kern, Mirels, and Hinshaw (1983), which showed that scientists do not consistently apply the rules of logic (specifically, *modus ponens*[8] and *modus tollens*[9]) in their reasoning, to argue that people are not as systematic as Fodor and

[7]Danny Sumina, personal communication.

[8]That is: $(P \rightarrow Q) \wedge P \vdash Q$, which reads, if P implies Q, and P is true then Q is also true.

[9]That is: $(P \rightarrow Q) \wedge \neg Q \vdash \neg P$, which reads, if P implies Q, and not Q is true then not P is also true.

1000	10
10	20
1000	30
20	40
1000	1000
30	1000
1000	1000
40	1000
(a)	(b)

Figure 2.3: A subject asked to perform a running additive total of the numbers listed in (a) may respond with 5000. When asked to perform the same task on the same numbers but in a different order (b) subjects are likely to respond with the correct answer 4100.

Pylyshyn claim. Most dramatically, of the 98 people examined only 41% correctly recognized the logical validity of modus tollens when reasoning about questions presented to them. Consequently, van Gelder and Niklasson argued that a Classical architecture is inadequate, as it fails to account for the degree of systematicity evident in the empirical studies. One could argue that the lack of perfect, or near-perfect accuracy is a competence/performance issue (i.e., the difference between what *can* be processed as distinct from what *is* processed in practice). However, the study also shows a statistically significant difference, in the percentage of scientists that recognized the logical validity of both *modus ponens* and *modus tollens*, between concrete and abstract examples. A classical model would predict no significant difference as concrete and abstract instances of, for example, *modus tollens* are structurally identical.

These three examples illustrate how some cognitive processes may operate purely on the basis of an associative or statistical relationship between two cognitive states, not on the basis of the structural content of those states. It is these forms of associatively- and statistically-based behaviours that the Classical framework does not address, and presumably, the sorts of behaviours that Connectionism would be well-placed to explain. The extent to which cognitive behaviour can be explained, however, is an empirical question[10], and not one addressed here.

The second limitation comes from the process-relative notion of tokening which precluded Connectionist architectures from being anything other than Classical implementations. The process-relative notion of tokening leaves open a vast class of architectures able to meet this condition. In an extreme case, tokening can be realized by a *look-up* table. For example, in the following systematicity of inference task, the first or second component is extracted from a binary relation. That is:

$$F((x_i, x_j), q_k) = \begin{cases} x_i & \text{if } k = 1 \\ x_j & \text{if } k = 2 \end{cases}$$

[10]Another empirical question is to what extent does the implementational level impact on the cognitive level. Charter and Oaksford (1990) and Butler (1993) have argued that the implementation level constrains the class of possible algorithms that can compute some cognitive function. Therefore, Connectionism at the implementational level may limit the choice of algorithms at the cognitive level.

where x_i and x_j are chosen from a set of N objects; and q_k is a query input requesting the first or second component. All $2N^2$ unique complex representations are generated by a hashing function:

$$f_H(x_i, x_j, q_k) = \#R, \tag{2.4}$$

where $\#R$ is a unique number for every unique triple (x_i, x_j, q_k). Now, the first and second constituents can be extracted by providing indexed functions:

$$f^1_{i,j}(\#R) = \begin{cases} x_i & \text{if } \#R = f_H(x_i, x_j, q_1) \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

$$f^2_{i,j}(\#R) = \begin{cases} x_j & \text{if } \#R = f_H(x_i, x_j, q_2) \\ 0 & \text{otherwise,} \end{cases} \tag{2.6}$$

where $f^1_{i,j}$ and $f^2_{i,j}$ are the class of function dedicated to recovering the first and second components, respectively. The results of the $2N^2$ access functions are then summed. Since, each function $f^k_{i,j}$ is sensitive to only one unique constituent, only one constituent will be returned.

In terms of network architecture, the internal representation can be physically realized as activation of a single hidden unit, and each access function as a connection from the hidden unit to another processing unit. The output of each processing unit is then summed at a single output unit (see Figure 2.4). The problem with this architecture is that it requires $2N^2$ processing units, and should be rejected on the same grounds as the microfeature solution was rejected. That is, on the basis that too many processing units are required.

However, it is not necessary to have dedicated one unit per object. Units can share the representation and processing responsibilities so as to minimize the total amount of resources. Auto-encoder networks of sigmoidal units[11], for example, require at most two hidden units to represent N objects (Kruglyak, 1990). By combining two auto-encoders, all N^2 binary relations can be represented with just four hidden units and $2N$ access functions. In this case, the number of processable

[11]That is, the activation function for the hidden and output units is: $f(net_i) = \frac{1}{1+e^{-net_i}}$, where net_i is the net input activation to unit i.

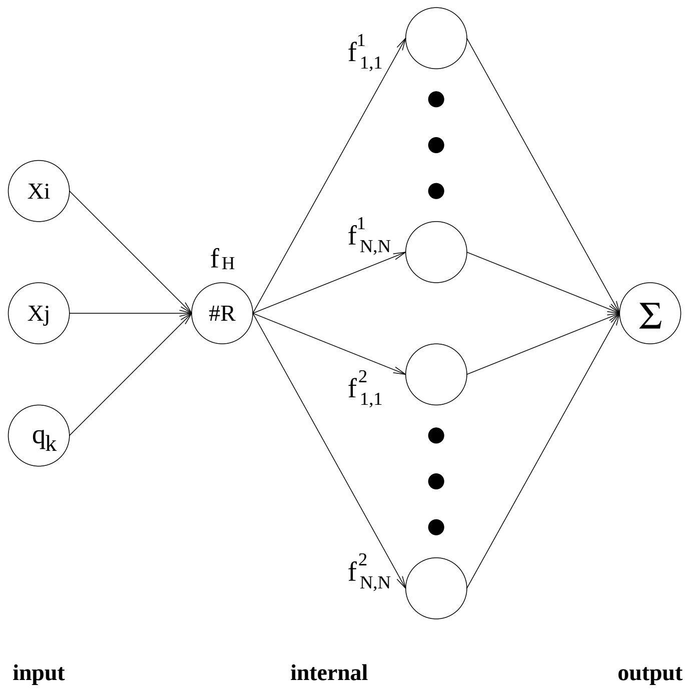


Figure 2.4: Connectionist implementation of a look-up table. Input units receive external representations x_i, x_j and q_k of the first component, second component and question, respectively. The activation function (f_H) for the first internal unit, whose activation is $\#R$, is given by equation 2.4. The activation functions for the second layer of internal units are given by equations 2.5 and 2.6. The activation from these units is summed by the output unit labeled $\sum$. Arrows indicate connections with a weight value of one.

objects grows faster than the amount of resource. Similarly, as suggested above, a network implementing Smolensky's tensor representation scheme could be provided with access functions at limited cost.

However, Fodor and McLaughlin rejected the possibility of a tensor (or other Connectionist) solutions on the grounds that it is not sufficient to show how systematicity is *possible* given the assumptions of a Connectionist architecture, one must also show how systematicity could be *necessary* given those assumptions. Since, in most cases, people exhibit systematicity despite their varying physical and social backgrounds. Fodor and McLaughlin argued that although it may be possible to hand-craft a Connectionist model that could represent the structured object *John loves Mary* given that it could represent the structurally related object *Mary loves John*, it is also possible to hand-craft a model that supports a representation of the structured object *Bill chased Sue* given that it supports the structurally unrelated object *John put the train in the box*. In short, even though Connectionist models could demonstrate systematicity by implementing Classical compositionality, Fodor and McLaughlin argued that they are too general to necessarily account for systematicity.

The weakness of Fodor and McLaughlin's argument, however, is that they do not specify criteria by which a (Connectionist) model is too general to necessarily account for systematicity. For example, a Connectionist architecture necessarily exhibits systematicity if the assumption is made that it be wired so as to exhibit systematicity. I attribute the lack of such criteria to be a consequence of Fodor and Pylyshyn's definition of systematicity being grounded, solely, in terms of computational capacity. That is, in terms of the desired functional behaviour of an architecture.

At this point, it is not clear whether the look-up table and tensor networks would be regarded by Fodor and McLaughlin as examples of tokening. It may be that the process-relative notion of tokening that I have described is too general for Fodor and McLauchlin. For on the one hand they entertained the possibility of non-Classical Connectionist models exhibiting systematicity by careful handcrafting of connections, but subsequently rejected such solutions on the basis that they do not necessitate systematicity. It is this lack of clarity that is precisely the problem one gets into when definitions are grounded in terms of computational capacity.

What is required is a tightening of the definition of systematicity along a dimension orthogonal to computational capacity. One such dimension often used in psychology is reaction time. Two computationally equivalent algorithms can be discriminated on the basis of the amount of time required to perform a task. Butler (1993) has used this dimension to argue for the importance of the implementational level.

Another dimension is learnability. In the domain of analogical reasoning, Halford and Wilson (1994) have attempted to reject a three-layer feedforward network model of the balance-scale task, also, on the basis of computational capacity alone. As I have argued in their case (see Appendix A), the learnability dimension can be used to discriminate two architectures on the basis of the amount of resource required to learn or acquire a function or behaviour. In the Classical framework, systematicity is built into a symbol system. However, Connectionism is concerned not only with the realization of cognitive behaviours, but also their acquisition. Thus, a "potential" contribution of Connectionism is in refining the characteristic properties of cognitive architecture on the basis of the learnability of systematic behaviour.

2.4 Thesis question

Connectionism is a general framework whereby global behaviours are a consequence of the interaction of simple units. Connectionist models are sufficiently general to represent any spatial function (Lapedes & Farber, 1988; Hornik, Stinchcombe, & White, 1989; Funahashi, 1989) and a general class of dynamic functions (Sato, Murakami, & Joe, 1990).

However, despite their capacity to realize any function there does not exist any general purpose learning algorithm except for restricted models. For example, the perceptron convergence rule (Hertz, Krogh, & Palmer, 1991) guarantees the perceptron will learn to discriminate between two classes of stimuli, provided that the stimuli (in their input representational space) are linearly separable (i.e., all points can be correctly partitioned by a single hyperplane). The limitation of the perceptron is that it cannot represent the function for discriminating points that are not linearly separable (Minsky & Papert, 1990). In fact, the general learning problem is known to be difficult computationally in the sense that it is unlikely that there exists a polynomial time algorithm for configuring weights of a network with three layers (Judd, 1987, 1990), even in the case where there is only one output and two hidden threshold units (Blum & Rivest, 1990, 1992).

This trade-off between the realizability of a range of functions/behaviours and their learnability is well-known from statistics as the bias/variance dilemma (Geman, Bienenstock, & Doursat, 1992). So, although Connectionist models have sufficient variance to approximate most behaviours, the important question, is: *do these models have sufficient bias to account for systematicity?*

Bias (in statistics), or innateness (in psychology) is a term that refers to the tendency for certain behaviour over others as a consequence of the architectural properties of the model. Architectural biases are both a boon and a burden for they may allow the tractable acquisition of one class of behaviours, but at the expense of being unable to exhibit another class. In the case of the perceptron, for example, restricting the unit activation function to be linear guarantees learnability of any linearly separable class of stimuli, but at the expense of not be able to discriminate non-linearly separable classes. Thus, use of a linear transfer function is an architectural bias. Conversely, use of a non-linear transfer function allows the discrimination of arbitrary classes, but does not guarantee learnability. A non-linear transfer function is also an architectural bias.

In a learning framework, where behaviour is (partly) acquired through experience, systematicity is a form of generalization over structurally related objects. Having learnt how to represent and process some instances of a complex object, a system can represent and process some instances of previously unseen, but structurally similar objects. It is the combinatorial nature of structured domains that makes the acquisition of systematic behaviours difficult for general purpose learning models like Connectionist networks. Even in the simple N-Vt-N example, if there are n agent/patients and m actions, then there are n^2m N-Vt-N complex objects. Simply allocating a new representation for each complex object would be infeasible on the basis that one is never likely to have previously seen all instances. Thus, an important question, and the primary question that this thesis attempts to address, is: *Can Connectionist models exhibit the necessary acquisition of systematic behaviour?*

2.4.1 Method

The acquisition of behaviour requires the necessary, but judicious, use of architectural biases. The question is, what are those biases in regard to the necessary acquisition of systematic behaviour. However, at the same time a particular bias should not preclude other desirable behaviours.

At this point the only property under consideration is systematicity. Having only one behaviour is one sided in that there is no cost. For example, systematicity could be trivially account for by direct implementation of a symbol system. However, this approach would be at the expense of the associative/statistical properties that potentially may explain some cognitive behaviours (e.g., the first option of Connectionism). Consequently, the approach taken here is to start from what Dyer (1991) calls "radical Connectionism". That is, from relatively unstructured general-purpose network architectures (e.g., feedforward and recurrent networks). This approach makes the fewest assumptions regarding a Connectionist architecture, and therefore precludes the fewest behaviours.

The other consideration is the criterion by which a network is regarded to have "necessarily" acquired systematicity. Since systematicity is regarded as a computational level property, such a criterion must be at this level and with respect to a particular task. In this thesis, two tasks are considered: (1) auto-association of N-tuples (as a means for capturing the degree of systematicity of representation of a model); and (2) extraction of a component from an N-tuple upon request (as a means for capturing the degree of systematicity of inference of a model). In conjunction with these two tasks are two degrees of generalization by which a model is considered systematic with respect to a task. They are: (1) systematicity as generalization over a domain of complex objects related by a common structure; and (2) systematicity as generalization across position: what Hadley (1993) calls *strong systematicity*. In contrast to Smolensky and van Gelder's approach, where systematicity was regarded primarily[12] as a representation problem, the approach taken here is to regard systematicity as a generalization problem.

At each investigation, the property that permits/prevents systematicity is analyzed. So, although this thesis is concerned with what properties give systematicity, this thesis is also concerned with *why* these properties do or don't give systematicity. The reason why a property of one architecture prevents systematicity is used as justification for moves to more structured architectures.

In summary, the approach taken in this thesis is as follows:

- specify a task for which it is possible to exhibit systematicity at the computational level;
- define a criterion for the necessary acquisition of systematic behaviour;
- evaluate a Connectionist model with respect to this task and criterion; and
- analyze the model in terms of the properties that permit/prevent systematicity.

2.5 Summary

Systematicity is a computational level property of cognitive systems whereby the ability to represent and process objects having a particular structure extends to

[12]Although Smolensky (1987b) also considers the possibility of learning an "optimal" combinations of role vectors, which is discussed in chapter 4.

other objects having the same structure. The importance of systematicity is that it places constraints on cognitive architectures.

Those constraints as argued for by Fodor and Pylyshyn are that cognitive architecture must consist of: (1) structured representations, or symbol structures; and (2) processes that are sensitive to the structure of those representations. These two constraints characterize a Classical cognitive architecture. The distinguishing feature of a Classical architecture is the tokening of constituents whenever complex representations are tokened.

Connectionists have proposed three alternative schemes of compositionality: Smolensky's weak (microfeatures) and strong (tensors) compositionality; and van Gelder's more general functional compositionality. After reviewing these alternatives, I have concluded that

- the Connectionist notions of compositionality either: cannot explain systematicity because the constituents are not uniquely accessible; or, implement Classical compositionality because the constituents are tokened (relative to their access processes) whenever complex representations are tokened.

Consequently, they fail to provide an alternative explanation of systematicity.

However, I have also argued that the limitation of the Classical position is that the requirement of systematicity is grounded, solely, in terms of computational capacity (i.e., what can be computed by an architecture). Consequently, systematicity can be realized by many computationally sufficient architectures, including Smolensky's tensors, which Fodor and McLaughlin wish to reject. I have suggested that the requirement of systematicity can be made stronger by considering learnability as providing an additional criterion for systematicity. Thus, a "potential" contribution of Connectionism is in an explanation for the necessary acquisition of systematic behaviour. Therefore, the problem that systematicity poses for Connectionism, which is the central concern of this thesis, is:

- Can Connectionist models exhibit the necessary acquisition of systematic behaviour?

The approach taken, in the subsequent chapters, in addressing this question is to: first, propose a suitable definition of the necessary acquisition of systematicity; and second, evaluate Connectionist models with respect to this definition; and in doing so ascertain the properties that permit/prevent the necessary acquisition of systematicity.

Chapter 3

Systematicity as generalization across domain

3.1 Introduction

In the last chapter I suggested that a "potential" import of Connectionism is in an explanation of the necessary acquisition of systematicity. This chapter is a first attempt at addressing the learnability of systematicity, rather than its computability. That is, where the debate reviewed in the previous chapter was concerned with algorithms/architectures that can compute systematic functions/behaviours, this chapter is concerned with algorithms/architectures that produce algorithms that exhibit systematic functions/behaviours.

The definition of systematicity from the previous chapter is a property whereby the ability to represent and process some instances of a structured object implies the ability to represent and process other instances with the same structure. In a learning framework then, systematicity is the property whereby the ability to learn to represent and process some instances of a structured object implies the ability to represent and process other instances with the same structure. In other words, systematicity is a form of generalization over structurally related objects. Some previous work has been done in this area by Brousse and Smolensky (1989) and Brousse (1991), and their work is re-examined in section 2.

In section 3, the framework of probably approximately correct (PAC)-learnability (Valiant, 1984) is presented, and used to provide the criterion by which a model is said to necessarily acquire systematic behaviour.

The classic criterion in computer science for which a problem is regarded as tractable (in terms of the amount of resource required), or for which an algorithm is regarded as a tractable solution to a problem is polynomial resource[1] complexity (Garey & Johnson, 1979). Algorithms, or more generally, problems which require more (exponential) resource are, generally, considered impractical as there are not enough resources to solve even small sized problems. Valiant extended the notion of tractability (feasibility) to the domain of learning. Briefly, a function (or, more generally, a function class) is considered learnable by an algorithm if the target function is acquired, to within some degree of accuracy on most occasions with only polynomial resources in the size of the target function. The PAC-learning framework has proven very useful in the formal analysis of learning and has led to a field within machine learning called *computational learning theory* (see Anthony, 1992, for an introduction to this field). In this chapter, the following definition is introduced: A Connectionist model is said to necessarily acquire systematic behaviour if

> *the architecture acquires the target behaviour to within a desired degree of accuracy, with a desired degree of confidence, with at most polynomial resources in the size of the behaviour.*

Feedforward networks are the first and perhaps the most widely used networks in Connectionist modeling. Given the work of Brousse and Smolensky they are an obvious starting point to addressing the issue of systematicity within Connectionism. In section 4, PAC-learnability is used as the criterion for the necessary acquisition of systematic behaviour with respect to the task studied by Brousse and Smolensky (i.e., auto-association of N-tuples - relations). Auto-association provides a way of evaluating whether a model (in this case, a feedforward network)

[1]Typically, resource means time, however, time can be traded for space, or number of processors in the case of parallel machines.

exhibits systematicity of representation.

General discussion regarding these results is given in section 5, and finally, a summary and conclusion are provided in section 6.

3.2 Brousse and Smolensky's *massive* generalization

In Connectionist models that learn from examples, cognitive behaviour is seen to emerge from the dynamics of the network interacting with some environment. One of the criticisms of the Connectionist approach has always been that Connectionist models require too many examples to acquire a competent level of performance in such complex domains as language. Combinatorial domains are interesting because such a vast amount of behaviour can be described with relatively few rules. In fact, they are the perfect domains for symbol systems, and the sorts of domains where a Connectionist model is expected to fail. It is this expectation of failure that motivated the studies of Brousse and Smolensky.

3.2.1 Task: Auto-association of N-tuples

One of the simplest combinatorial domains is the auto-association of N-tuples, where say, an object represented as N instantiated variables, must be encoded and later recovered. The set of N-tuples S is defined (using Z-notation Spivey, 1989) as: $S = S_1 \times \ldots \times S_N = \{x_1 : S_1; \ldots ; x_N : S_N \bullet (x_1, \ldots, x_N)\}$, where S_i is the set of possible values at position i of the tuple. The case considered here is where $S_1 = S_2 = \ldots = S_N$. For example, (x_3, x_4) and (x_4, x_3) are distinct objects in a domain of 2-tuples. The auto-association of N-tuples is the identity function that maps every element in S to itself.

3.2.2 Method

Brousse and Smolensky (1989) and Brousse (1991) studied the auto-association of N-tuples (strings of length N) by a feedforward network. In their simulations, the number of atomic objects was 26 (letters of the alphabet), and the tuple order was varied from 2 to 6 (i.e., the number of components in each tuple). Therefore, the size of the domain grew exponentially with the tuple order from $26^2 = 676$ ($N = 2$) to $26^6 \approx 3 \times 10^8$ ($N = 6$).

Each atomic object (letter) was represented externally (i.e., at the input/output layer) by a random binary vector of length 8. The input/output representation of an N-tuple (string) is the in-order concatenation of the vector representations of its components. Thus, the representation of an N-tuple is a $8N$ dimensional vector.

The feedforward network they examined consisted of $8N$ input units (i.e., one unit for each vector component) completely connected to all $5N$ hidden units, which are in turn completely connected to all $8N$ output units. The task of the network is to learn to construct internal (hidden unit vector) representations of the input, and subsequently, recover the input representation at the output (e.g., Figure 3.1).

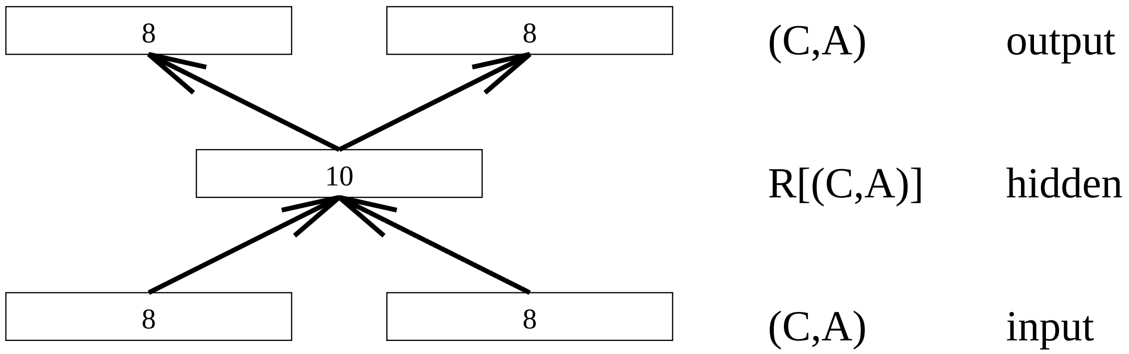


Figure 3.1: A feedforward network for learning to represent 2-tuples. Arrows indicate complete connections between groups of units, and numerals indicate the number of units in a group. In this example, an input representation of the object (C, A) is mapped to an internal representation $R[(C, A)]$, from which the input representation is recovered at the output.

A subset of the domain is randomly selected from a uniform distribution on which the network is trained. Training consists of presenting the input and target vector pairs. The network learns by changing the weights of connection so as to minimize the error between the network's output and the target output for each pattern using the standard backpropagation learning algorithm (Rumelhart et al., 1986), which is defined as:

$$\Delta W_{ij} = -k \frac{\partial E}{\partial W_{ij}} \tag{3.1}$$

where ΔW_{ij} is the change in the weight of the connection from unit i to unit j; k is the magnitude of change; and E is the network error, which, in this case, is the sum of the squared differences between the target vector and the network output vector, given by:

$$E = \sum_{p=1}^{n} E_p = \sum_{p=1}^{n} |\vec{t}_p - \vec{o}_p|^2 = \sum_{p=1}^{n} \sum_{i=1}^{m} |t_{pi} - o_{pi}|^2 \tag{3.2}$$

where E_p is the error for each example pattern (p); n is the number of examples; $\vec{t}_p$ and $\vec{o}_p$ are the target and network output vectors, respectively; and t_{pi} and o_{pi} are the corresponding vector components at output unit i for m output units. The activation of any non-input unit is given by:

$$o_j = \frac{1}{1 + e^{-\sum_{i=1}^{k} o_i w_{ij} + b_j}} \tag{3.3}$$

where o_j is the activation of unit j; o_i is the activation of unit i whose activity is propagated to unit j via a connection with weight w_{ij}; k is the number of units that have an incoming connection to unit j; and b_j is the bias.

Training was started from a random setting of weights and terminated when all output units were within 0.4 of their targets for all patterns.

Testing the performance of the network consists of randomly resampling the domain and determining whether the network's output is sufficiently close to the target output. In this case, the network's response was considered correct when all output units are within 0.4 of their corresponding target values.

3.2.3 Exponential number of generalizations

From this training scheme, Brousse and Smolensky showed that from a fixed number of training examples (50), the number of generalizations (future examples correctly processed) and virtual memories (patterns learnt within five additional learning trials) grew exponentially with the order of the tuple (see Brousse & Smolensky, 1989, Figures 5 & 7, respectively).

3.2.4 Exponential decrease in percentage of generalizations

The exponential growth in number of generalizations exhibited by the feedforward network appears impressive. However, the growth in the total number of examples is also exponential. Two measures of the network's degree of systematicity for the representations of tuples can be obtained by calculating (1) the number of generalizations; and (2) the number of generalizations plus virtual memories, (from Brousse & Smolensky, 1989, Figures 5 & 7, respectively) as a percentage of the total number of examples in the domain.

A plot of tuple order versus degree of systematicity is shown in Figure 3.2. The linear relationship between the x and y axes of the log-linear plot indicates a decreasing exponential relationship between tuple order and degree of systematicity. So, for example, despite the network being able to represent tuple instances of order 6, on average only 1% were correctly represented, considering the number of generalizations plus virtual memories.

It is this lack of generalization over such structured domains (i.e., lack of systematic behaviour) that has been the target of the strong criticism of the Connectionist approach to cognitive modeling (Fodor & Pylyshyn, 1988). Brousse (1991) used a weight decay term[2] to improve generalization to the point where the network performed perfectly for small N. However, the percentage of correct examples still decreased as an exponential[3] function of N. Clearly, 50 training examples is too few to achieve a high degree of generalization over such a large domain. However, if the network requires some constant (even if small) fraction of the domain then learning will be intractable as the training set size will grow exponentially with

[2]Weights are eliminated so as to reduce the possibility of overparameterization by the network.

[3]Brousse (page 190) reports the number of generalizations to be 15^N. However, the total number of examples is 26^N. Consequently, the fraction of example space correctly processed is $(\frac{15}{26})^N$ (i.e., exponentially decreasing function of N).

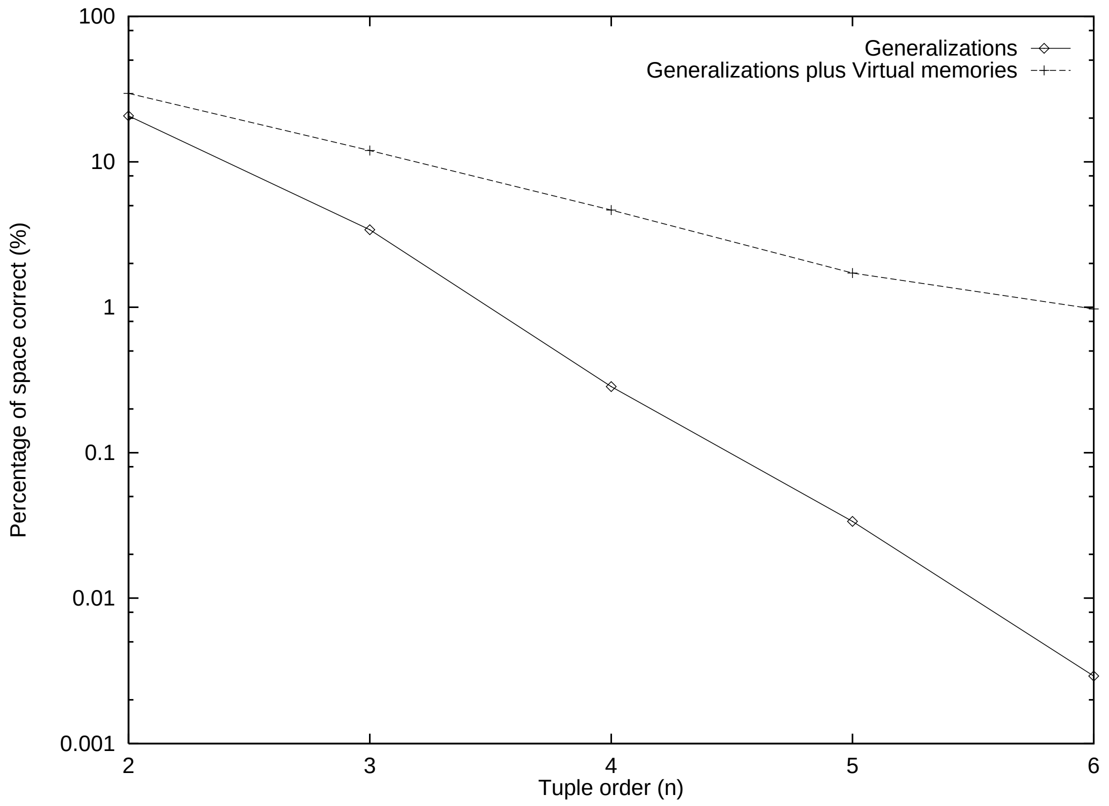


Figure 3.2: A log-linear plot of the number of generalized and generalized plus virtual memories as a function of tuple order (N). The linear relationship indicates an exponential decrease in percentage of space correct. Reconstructed from Brousse and Smolensky (1989); Figures 5 & 7.

tuple order.

If systematic behaviour is regarded as a degree of generalization over a particular domain, then the problem can be restated in terms of PAC-learnability by asking two questions. First, how many training examples are required to maintain a fixed degree of accuracy over the entire space? (In other words, what is the *sample complexity*?) Second, how long will it take to learn these examples? (In other words, what is the *time complexity*?)

Before attempting to answer these two questions, the concept of PAC-learnability is explained with emphasis on its applicability to the systematicity problem.

3.3 Probably approximately correct learnability

Probably approximately correct (PAC)-learnability is a theoretical concept introduced by Valiant (1984) as a criterion against which one may formally prove the learnability of a particular class of problems by a particular learning machine.

The concept of PAC-learnability is analogous to, and extended from the concept of tractability. Roughly, an algorithm is said to tractably compute some function if it requires no more than a polynomial amount of resource (e.g., time, space, processors) in some parameter measuring the size of the function. Similarly, a function (or function class) is said to be PAC-learnable by an algorithm if that algorithm, on most occasions, acquires the target function to within a desired degree of accuracy with no more than a polynomial amount of resource in the size of the target function.

PAC-learnability has been used as the framework for theoretical results on the learning capabilities of neural networks. (See Anthony, 1992, for a review of the import of computational learning theory to Connectionism). Theoretical approaches to learnability problems using neural networks, in general, divide the problem into two parts: sample complexity - the number of training examples required to acquire the target function; and time complexity - the time required to *load* (correctly map) these training examples.

In determining the sample complexity, one wants to establish how many training examples are required before the network correctly processes a new example with probability greater than $1-\epsilon$. Furthermore, the probability that the network will behave with this degree of accuracy on future examples having been given this many training examples must be greater than $1-\delta$. A network is considered to tractably learn a function when the number of training examples required to satisfy that relationship is a polynomial of the size of the target function, which is usually measured as the number of weights required to implement that function. A number of results on the sample complexity of neural networks have been established (e.g., Baum & Wilczek, 1987; Shawe-Taylor & Anthony, 1991; Haussler, 1992), and it is one of the results of Shawe-Taylor and Anthony that is used in the next section.

Establishing time complexity results generally involves mapping the task of finding a set of weights satisfying the function specified by the training set for a given network architecture into a problem with known complexity in computability theory (e.g., Judd, 1987, 1988; Lin & Vitter, 1989; Blum & Rivest, 1990; Judd, 1990; Blum & Rivest, 1992). These results, although general, are also negative, in the sense that, even for relatively simple feedforward architectures there does not exist a polynomial time algorithm for finding a set of weights that will implement an arbitrary function specified by the training set.

However, these results are worst case scenarios in that they consider the time required to learn any function from very general function classes. Time complexity can be reduced by restricting the function class, or by incorporating knowledge regarding the functions to be learnt, into the learning algorithm. Since learning is essentially a search through a function space for a function that is sufficiently close to the target function, restricting the function class effectively reduces the search space and therefore the search time. Incorporating knowledge about the functions to be learnt structures the search space into regions where target functions are likely to be found. Consequently, search time can be reduced by ignoring regions of the search space, just as is the case, for example, when performing a binary search.

Incorporating domain-specific features into an algorithm or architecture (e.g., non-linear activation function) can, however, greatly complicate a theoretical approach. Consequently, in the next section, the time complexity is determined empirically.

PAC-learnability provides a tool for determining the learning capabilities for such potential cognitive models as Connectionist networks. If one links the desired degree of generalization with the computational resources required to achieve that degree of generalization then one has a criterion for determining whether a Connectionist model is systematic that does not rely on the computational capacity of the model. The rationale for using polynomial resources as a criterion for systematicity is that, if a machine or model requires a greater amount of resources than polynomial (e.g., exponential), then there simply isn't enough time or resource available to achieve the desired level of generalization for even moderately sized functions. Thus, this criterion allows potential candidates for cognitive architecture to be discounted on the basis that they are too computationally expensive.

In the next section, this criterion is applied to the task studied by Brousse and Smolensky to examine the capacity of feedforward networks to acquire systematicity of representation.

3.4 Systematicity of representation with feedforward networks

Systematicity of representation is the ability to construct internal representations of structurally related objects. For these internal representations to be *useful* the objects they represent must be subsequently recoverable.

For example, a model that constructs internal (vector) representations of complex objects by the addition of vector representations of the object's components would not be satisfactory since some objects would not have unique internal representations. Suppose, for example, internal representations of the two distinct objects *John loves Mary* and *Mary loves John* were composed by vector addition of their components *John*, *Mary* and *loves*. Then:

$$\begin{aligned}
\vec{R}_{John\ loves\ Mary} &= \vec{R}_{John} + \vec{R}_{loves} + \vec{R}_{Mary} && \text{by definition} \\
&= \vec{R}_{Mary} + \vec{R}_{loves} + \vec{R}_{John} && \text{by law of commutativity} \\
&= \vec{R}_{Mary\ loves\ John} && \text{by definition}
\end{aligned}$$

In fact, any operator with the property of commutativity is an inadequate mechanism for the construction of complex representations.

The usefulness of the internal representations constructed by a network can be tested by requiring the network to subsequently output a representation of the complex object it was presented. In other words, by requiring the network to auto-associate the input representations of complex objects.

3.4.1 Task: Auto-association of N-tuples

The family of tasks on which the network is evaluated is the same as used in the studies by Brousse and Smolensky as described in subsection 3.2.1. That is, auto-association of N-tuples where an N-tuple is an ordered relation of N atomic (unstructured) objects selected from some set.

3.4.2 Model: Feedforward network

The feedforward network (Figure 3.3) used in these simulations is the same as used by Brousse and Smolensky except that the number of input/output units per tuple order was 10 (i.e., $k = 10$) and tuple order (N) was varied from 2 to 10, whereas in the work of Brousse and Smolensky, $k = 8$ and N was varied from 2 to 6. Also, a different input and output encoding scheme was used. Each component was encoded by a *local* encoding scheme (i.e., one unit *on* and the rest *off*) at the input layer and a *block* encoding scheme (i.e., half of the units *on* and the rest *off* Bakker, Phillips & Wiles, 1993; in press) at the output layer (see section 3.4.4), whereas

Brousse and Smolensky encoded each component as a random binary vector. With local and block encoding schemes, the number of possible components is equal to the number of units available for representing each component, which in this case is 10.

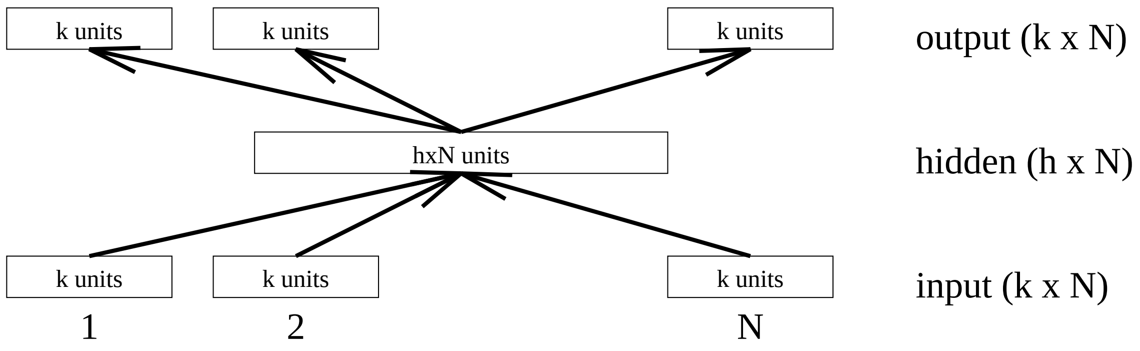


Figure 3.3: Feedforward network for the auto-association of N-tuples. In this architecture k refers to the number of possible atomic objects in any one position, N refers to the number of components within a complex object (tuple order), and h refers to the number of hidden units per component. Every input unit was connected to every hidden unit, which was connected to every output unit.

In this chapter, systematicity has been framed in terms of PAC-learnability. That is, a network is considered systematic if it can attain a fixed degree of generalization (i.e., percentage of example space) by requiring, at most, polynomial resources in the size of the function. For the task used here, the size of the function is characterized as the tuple order (N).

Here, both a theoretical and an empirical approach is taken to address this issue of computational resource. A theoretical result places bounds on the difficulty of the problem. By using a result due to Shawe-Taylor and Anthony (1991) an upper bound on the number of training examples needed to attain a desired degree of generalization can be obtained. The advantage of this approach is that one can obtain limits purely on the information contained in the data and the representational power of the model which are independent of the learning algorithm the model may use. The limitation of this approach, however, is that there are no general results for determining time complexity. Therefore, the time complexity results were established empirically.

3.4.3 Theoretical result

Firstly, a theoretical upper bound on sample complexity is given for this task. Then, the sample and time complexities are determined empirically.

Polynomial sample complexity

The sample complexity result of Baum and Haussler (Baum & Haussler, 1989) is not applicable in this case as multiple output units are involved. However, there is a corollary by Shawe-Taylor and Anthony (Shawe-Taylor & Anthony, 1991) that, interestingly enough, provides an upper bound on the number of training examples independent of the number of output units. It applies to feedforward networks of threshold units with one hidden layer. It says:

> "Given an accuracy parameter ϵ and a confidence parameter δ, for a feedforward network with W variable weights and n computational nodes, with probability greater than $1-\delta$ the network will give correct output with probability greater than $1-\epsilon$ on inputs drawn according to some distribution, provided it correctly computes the function on a sample (drawn from the same distribution) of size at least
>
> $$m_0 = m_0(\epsilon,\delta) = \frac{1}{\epsilon(1-\sqrt{\epsilon})}\left[ln\left(\frac{1.3}{\delta}\right) + 4(W+n)log_2(en)ln\left(\frac{6}{\epsilon}\right)\right]$$
>
> (Shawe-Taylor and Anthony, 1991: p116)"

where W is the total number of variable weights (including biases) of connections from input units to hidden units and hidden units to a single output unit[4]; and n is the number of hidden units. In a network of N $(k-h-k)$ encoders, $n = N \times h$, and $W = khN^2 + 2hN + 1$.

If the accuracy and confidence parameters are fixed, as well as the number of possible values at each tuple position (k), then an upper bound on sample complexity in terms of N, the tuple order, can be determined. Given that the number

[4]Since their result is independent of the number of output units, W is specified so as to consider only weights associated with one of the output units.

of input and output units is $N \times k$, and the number of hidden units[5] is $N \times \lceil log_2 k \rceil$, then the upper bound on the maximum number of training examples required is

$$m_0 = O\left(\frac{N^2 k\, log_2 k}{\epsilon} log_2 \left(\frac{N\, log_2 k}{\epsilon}\right)\right) = O(N^2 log_2 N)$$

Thus, the network only requires a polynomial number of training examples. Provided the network can *load* (acquire the correct behaviour on) these examples it will, with high probability, correctly generalize to an exponential number of future examples.

3.4.4 Empirical results

The theoretical result is quite general in that it applies to any function that is representable by such a network (not only auto-association of N-tuples), and it is independent of the distribution from which the training and testing examples are selected (it only requires that the training and testing distributions be the same). However, the bound put on the number of training examples is quite high (in the order of tens of thousands of training examples). Furthermore, there is an assumption that there exists an algorithm that can load these examples in polynomial time.

With respect to the auto-association of N-tuples task, the question is: Can the standard backpropagation algorithm be used to load enough training examples in polynomial time so that the feedforward network exhibits systematicity with respect to this task?

Method

In an empirical study, it is not necessary to separate complexity into sample and time components. One could simply record the total processing time required by the network to acquire the desired degree of generalization and subsume sample complexity within time complexity.

[5]The minimum number of threshold units required by a single encoder of k items is $\lceil log_2 k \rceil$, using a binary encoding scheme.

However, keeping the two components separate allows one to determine where the demand for computational resource is greatest, and therefore it allows one to identify the weaknesses of the architecture. For example, it is possible that the sample complexity is low relative to the time complexity to load the examples. Such a result suggests that the learning algorithm used by the network is not making efficient use of the information represented in the training set. It would suggest that improvements are to be found in the learning algorithm rather than the connectivity of the network. This point will be returned to when discussing the empirical results of the feedforward network on the auto-association of N-tuples task.

For the empirical approach, what is considered is the number of training examples and the time taken to obtain at least 95% accuracy ($\epsilon = 0.05$) with at least 99% confidence ($\delta = 0.01$) on future examples for tuples of order $N = 2$ to $N = 10$ (in increments of 2). The number of possible values at each tuple position was set at 10, as mentioned in subsection 3.4.2 (i.e., $|S_i| = k = 10$).

Each network was trained using the standard backpropagation algorithm[6] (Rumelhart et al., 1986) on training sets ranging from 10 to 1500 examples randomly generated from a uniform distribution. A train-test trial consisted of: random initialization of network weights; random generation of training examples; training the network until all output units were on the right side of 0.5 for all training patterns; testing network performance on a new test set of 1000 randomly generated examples from the same distribution; and recording the number of test patterns for which all output unit activations were on the right side of 0.5.

For each value of N and each training set size, five train-test trials were conducted, providing five measurements of network performance on the 1000 test examples, from which the 99% confidence intervals on network performance on future examples could be calculated. For each N, training set size was increased until the lower bound of the 99% confidence interval was at least 95%. In other words, the training set size was increased until any one of the randomly initialized networks,

[6]Which is the same algorithm as used by Brousse and Smolensky (see subsection 3.2.2).

with 99% confidence, would correctly process at least 95% of the 1000 test examples it was given. The training set size which met this criterion was recorded. The time taken to train each network, measured in total number of weight updates, was also recorded.

In this simulation, sigmoidal units (as defined by equation 3.3) rather than thresholds were used as it reduces the number of necessary hidden units[7] to $2N$. The learning rate was set to 0.1. No momentum term was used. Weights were updated after each training pattern was presented. Weights were randomly initialized from a uniform distribution in the range $[-1, 1]$. The learning algorithm used was gradient descent with the sum of squares error function (as defined by equations 3.1 and 3.2, respectively).

The representation of each component at each group of k units at the input layer was a local encoding (i.e., one unit *on* and the rest *off*), and at the output layer was a block encoding (i.e., half of the units are *on* consecutively, with wraparound, and the rest *off*). That is,

$$\begin{array}{rcll} & & \textbf{Input (local)} & \textbf{Output (block)} \\ x_1 & = & 1\,0\,0\,0\,0\,0\,0\,0\,0\,0 & 1\,1\,1\,1\,1\,0\,0\,0\,0\,0 \\ x_2 & = & 0\,1\,0\,0\,0\,0\,0\,0\,0\,0 & 0\,1\,1\,1\,1\,1\,0\,0\,0\,0 \\ \vdots & & \vdots & \vdots \\ x_{10} & = & 0\,0\,0\,0\,0\,0\,0\,0\,0\,1 & 1\,1\,1\,1\,0\,0\,0\,0\,0\,1 \end{array}$$

The block encoding scheme for the output layer was chosen as it dramatically improves the learning time of the $k-2-k$ encoder problem (Bakker, Phillips & Wiles, 1993; in press), which has previously been shown to be particularly difficult for backpropagation-style networks (Lister, 1993).

Result(1): Polynomial sample complexity

The number of training examples, sufficient to meet the 99% confidence level (i.e., $\delta = 0.01$) and 95% accuracy level (i.e., $\epsilon = 0.05$) for tuple orders ranging from 2 to

[7]Two hidden units being the most required for a single encoder of sigmoidal units (Kruglyak, 1990).

10 is shown in Figure 3.4.

On a log-log plot, the linear relationship between the x and y coordinates (solid line) indicates that training set size is a polynomial function of tuple order, whereas the total number of examples is an exponential function of tuple order and appears as a non-linear relationship between the x and y coordinates (dotted line).

A linear regression shows that the relationship between number of training examples (m) and tuple order (N) is:

$$m = 130\ N^{1.01}$$

with significance $F(1,3) = 44$, $p < 0.01$.

Thus, the sample complexity of the feedforward network on the auto-association of N-tuples task is polynomial in the tuple order.

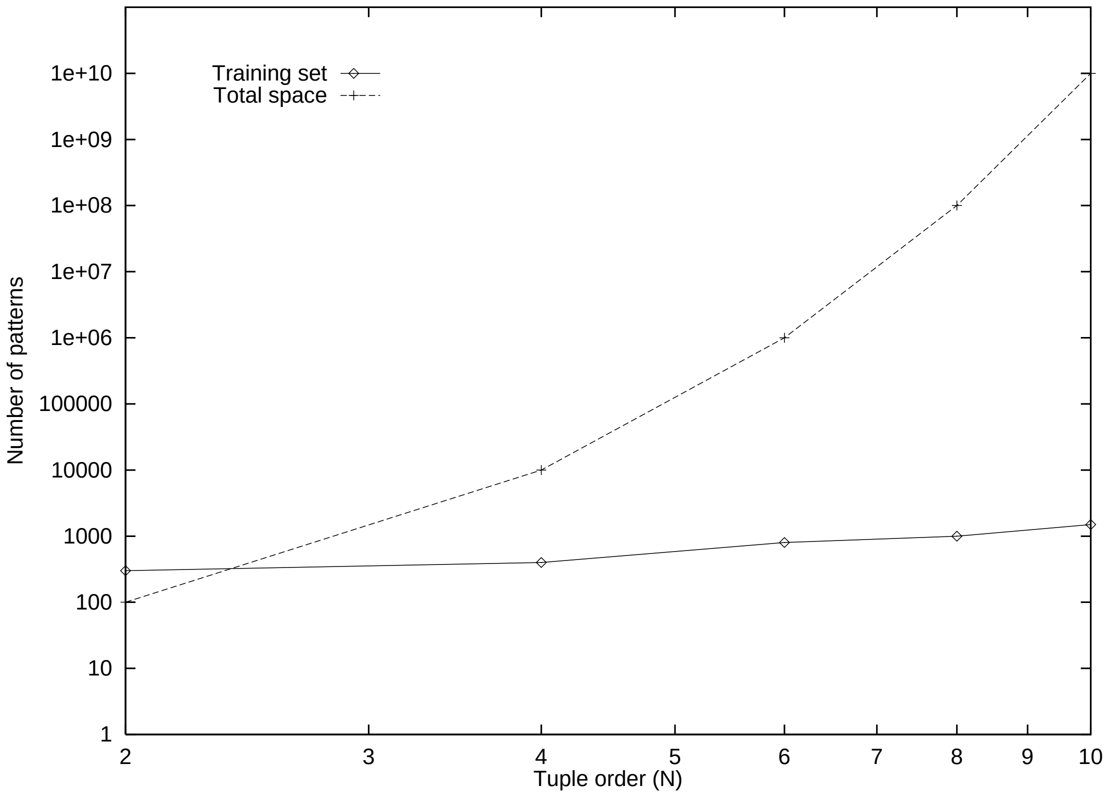


Figure 3.4: A log-log plot of the number of training examples (solid line) and total possible examples (dashed line) as a function of tuple order (N). The linear relationship indicates that the required number of training examples is a polynomial function of tuple order (i.e., sample complexity is polynomial).

Result(2): Polynomial time complexity

The mean training time, measured as the total number of weight updates (over 5 trials), sufficient to load the training examples for tuple orders ranging from 2 to 10 is shown in Figure 3.5.

On a log-log plot, the linear relationship between the x and y coordinates indicates that training time is a polynomial function of tuple order.

A linear regression shows that the relationship between training time (t) and tuple order (N) is:

$$t = 2.9 \times 10^8 \; N^{5.1}$$

with significance $F(1,3) = 1020$, $p < 0.01$.

Thus, the time complexity of the feedforward network on the auto-association of N-tuples task is polynomial in the tuple order.

Discussion

The theoretical result assumed the existence of an algorithm that could load the training examples in polynomial time. The question was whether the standard backpropagation algorithm could be used to load enough training examples in polynomial time so that the feedforward network would demonstrate the necessary acquisition of systematicity of representation with respect to the domain of N-tuples. For the empirical studies, the feedforward network is said to necessarily acquire systematicity of representation if, on 99% of occasions, it acquires the capacity to represent N-tuples with accuracy greater than or equal to 95%, requiring computational time that is at most a polynomial of N. The empirical results showed that the feedforward network exhibited the necessary acquisition of systematicity as defined by this criterion.

The number of training examples required was almost linear ($O(N^{1.01})$) in the tuple order. The total time required to load these examples was roughly a quintic ($O(N^{5.01})$) in the tuple order. The time complexity can be reduced to $O(N^3)$ if one considers that there are $O(N^2)$ weights that can be updated in parallel.

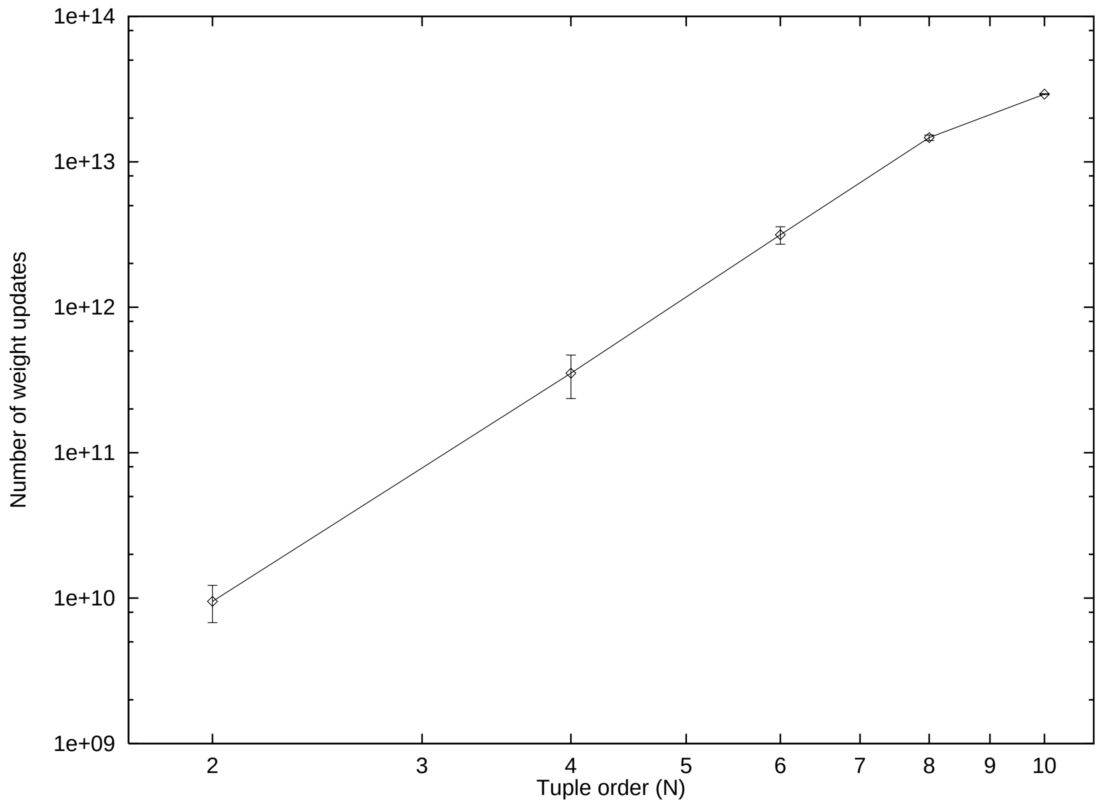


Figure 3.5: A log-log plot of the mean number of total weight updates (over 5 trials) as a function of tuple order (N). Error bars indicate one standard deviation. The linear relationship indicates training time as a polynomial function of tuple order (i.e., time complexity is polynomial).

By separating resource requirements into sample and time complexity components one can see that the network did not require many examples to learn the function. The expense was in learning these examples. Consequently, greatest reductions in resource requirements would be found with improvements to learning. One possibility is the use of an exponential error function (see, for example, Baum & Wilczek, 1987; Lister, Bakker, & Wiles, 1993), instead of the quadratic error function used here (i.e., equation 3.2).

The number of generalizations exhibited by the network was very high. For example, at $N = 10$, the network's degree of generalization was estimated to be 95% of 10^{10} test patterns[8], from approximately 1500 training patterns. The high degree of generalization was due to the small number of weights needed to represent the function. Sample complexity was proportional to the number of weights. Restricting the number of weights (by restricting the number of hidden units) reduces the class of functions that can be represented by the network. Less information (in the form of training examples) is needed to distinguish the target function from the class of representable functions as there are fewer representable functions. In the feedforward network, the number of weights was a polynomial function of tuple order. If the number of weights were to increase as an exponential function of tuple order then, by Shawe-Tayler and Anthony's result, the upper-bound on number of training examples would also increase as an exponential function. This assumption would make systematicity, as defined in terms of PAC-learnability, not achievable. Thus, an architectural assumption required in the feedforward network is that the number of weights grows as a polynomial function of tuple order.

The use of a block encoding scheme was important in establishing the polynomial time complexity result. Bakker et al. (1994) showed that, for a single auto-associator (i.e., $N = 1$), training time as a function of the number of possible components (k) grew rapidly when outputs were encoded locally, yet slowly when using block encoding. For example, at $k = 9$ ($N = 1$), the number of pattern

[8]Since the 1000 test patterns were randomly generated from a uniform distribution for each train-test trial.

presentations[9] was approximately 7×10^5 for local encoding compared to approximately 4000 for block encoding. For the simulations performed here, it was not possible to get the network to learn to correctly map all training examples under a local encoding scheme, and so block encoding of the outputs was used.

Use of block encoding at the outputs changes the organization of internal representations and hyperplanes at the hidden layer of the network. Lister (1993) and Bakker et al. (1994) showed that, after training a network on 1-tuple auto-associations where the output patterns were encoded locally, the hidden unit vector representations were arranged in a circle with the hyperplanes (i.e., the decision boundaries of the output units) separating one point from all other points. Figure 3.6(a) characterizes this organization for local encoding. Bakker et al. (1994) showed that in the case of block encoding hyperplanes were oriented so as to bisect the organization of hidden unit vector representations. Figure 3.6(b) characterizes this organization for block encoding. The change in internal organization affects the error surface which determines learning time. Effectively, each hyperplane is performing a classification. In the case of local encoding, the distribution of class members is skewed into two classes of 1 and $N - 1$ points, respectively. In the case of block encoding, it is relatively balanced, with about $\frac{N}{2}$ points in each class. The effect of these two distributions on error surface was examined and discussed in Phillips (1993). Essentially, a block encoding changes the error surface so as to provide more information as to the location, in weight space, of the target function (Bakker et al., 1994). More information is provided in the sense that more of the error surface points (in terms of direction of steepest descent) to the region containing the target function. Thus, the search time is effectively reduced by ignoring more of the search space.

Given the architectural assumptions of: complete connectivity between layers; polynomial growth in the number of variably weighted connections; block encoding of objects at the output layer; and a set of training and testing examples that were selected from the same uniform distribution the feedforward network demonstrated

[9]One pattern presentation involves a forward propagation of activity followed by a backpropagation of error, after which weights are updated.

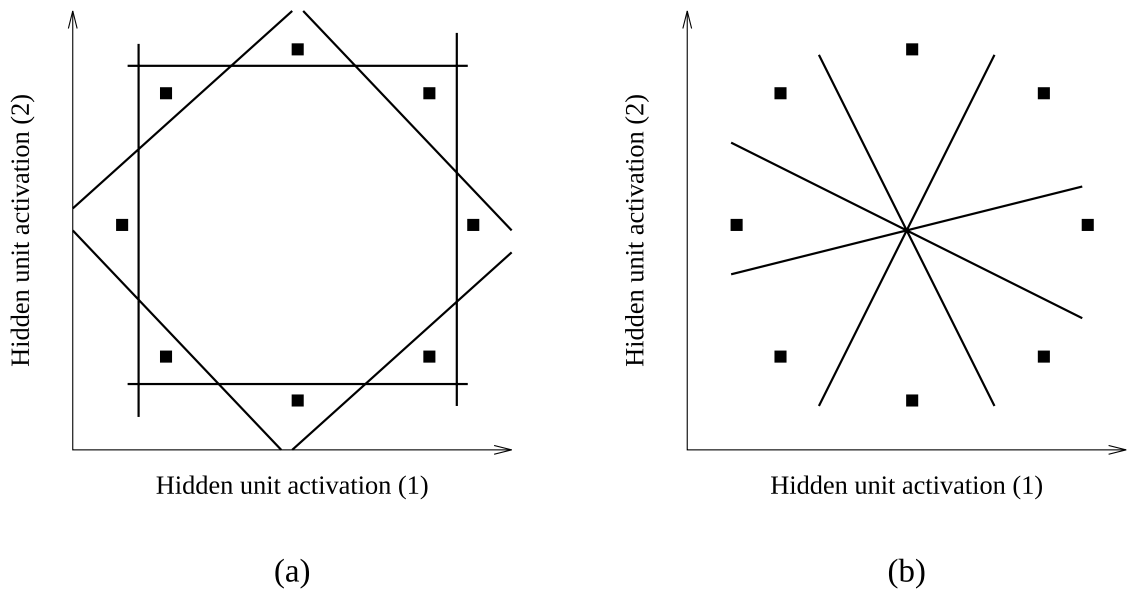

Hidden unit activation (2)
Hidden unit activation (1)
(a)
Hidden unit activation (2)
Hidden unit activation (1)
(b)

Figure 3.6: Hyperplane orientation for an 8-2-8 encoder. Characteristic orientation of hyperplanes (dark solid lines), and hidden unit vector representations (solid squares) for local output encoding (a) and block output encoding (b) for 1-tuple auto-association of 8 possible components (i.e., $N = 1$, $k = 8$ and $h = 2$).

the necessary acquisition of systematicity with respect to the auto-association of N-tuples task. The criterion for the necessary acquisition of systematicity was that the time required to acquire the target function to within 95% accuracy with 99% confidence was no more than a polynomial function of tuple order.

3.5 General Discussion

Having presented a case for a Connectionist model that necessarily exhibits systematicity the next question to ask is whether such a demonstration addresses Fodor and McLaughlin's concerns regarding the necessity for systematicity given the assumptions of the architecture. There are three areas in which this demonstration could be criticized.

- The assumptions made in the architecture were too strong. These assumptions were:
    - polynomial growth in the number of variably weighted connections;
    - complete connectivity between layers;

  - block encoding of representations at the output layer.

- The criterion for the necessary acquisition of systematicity is too weak.

- The demonstration of the feedforward network in meeting this criterion was too narrow.

In this section, each of these point are discussed in turn.

A polynomial growth in number of connections would appear to be a reasonable assumption. An architecture requiring more resources (e.g., exponential) soon becomes unrealistic with even small sized functions. It is this criterion that computer scientists use to separate feasible and infeasible algorithms. However, Halford et al. (1994) have argued that an assumption of exponential resource is reasonable if one considers that the order of relations that humans typically can process is limited to 4 or 5. Furthermore, an architecture that required exponential resources would explain these processing capacity limitations.

The assumption of complete connectivity is probably unrealistic given that it is generally known that real neurons are not completely connected. However, the theoretical result is a statement about the number of weights, not the connectivity of the network, and so still applies. In the empirical case, the network relied on complete connectivity between layers since the number of hidden units to represent every tuple instance was minimal[10]. This restriction could be relaxed by using more hidden units. An increase in hidden units would increase the training time, but it is not expected to be more than polynomial provided the number of hidden units is not more than polynomial.

The strongest assumption is block encoding. Block encoding imposes *a priori* an ordering, or similarity on component representations. For example, the vector **1 1 1 0 0 0** is closer to the vector **0 1 1 1 0 0** than the vector **0 0 1 1 1 0** under the euclidean distance metric. However, in the previous chapter it was mentioned that systematicity is a property at the level of compositional objects, not at the level of atomic objects. Thus, there should be no *a priori* similarity between components.

[10]The minimum number of hidden units for 1-tuple auto-association is two (Phatak, 1993).

In the subsequent chapters, the assumption of external representation is weakened by using local encoding for output representations.

In the debate over the import of systematicity no quantification of empirical evidence was given to support or refute Connectionist models. Thus, there are no absolute levels of generalization and confidence with which to test models. The importance of the theoretical result is that for any specified level of accuracy and confidence, the sample complexity is a polynomial function of the tuple order. The accuracy and confidence levels chosen for the empirical studies were, although somewhat arbitrary, chosen to show that the network could attain a high degree of generalization with a high degree of confidence. It is possible that people acquire systematic behaviour, to whatever absolute levels, with fewer than polynomial examples. In this case, a tighter criterion for systematicity is required. In the next chapter, the feedforward network is re-examined on a different criterion based on linguistic evidence.

A cognitive architecture must not only be able to represent complex objects, but also make inferences from them. For example, a cognitive architecture must be able to infer *Sue* from the sentence-question pair *Sue went to the store. Who went to the store?* (i.e., the architecture must be capable of extracting, on request components from complex objects). In the next chapter, the feedforward network is re-examined on a systematicity of inference task, as well as a systematicity of representation task. The domain of N-tuples was chosen because the number of complex objects grows exponentially with N. Given the work of Halford et al. (1994) it may be more appropriate to examine growth in terms of number of possible components with tuple order fixed (i.e., training time as a function of k), as was done in (Wiles, Phillips, & Norris, 1993). With a PAC-learnability criterion, however, this would not be an interesting case as the total number of objects in the domain grows only as a polynomial of k. Thus, perfect systematicity can be demonstrated by simply memorizing all objects.

3.6 Summary and conclusion

In this chapter, the necessary acquisition of systematicity has been defined in terms of PAC-learnability. That is, a network is said to necessarily acquire a systematic behaviour if, with a high degree of confidence, the time required to learn that behaviour to a high degree of accuracy is no worse than a polynomial function of a parameter that measures the size or complexity of that behaviour.

Previous work by Brousse and Smolensky (1989) and Brousse (1991), which showed an exponential number of generalizations with respect to tuple order (N) in an auto-association of N-tuples task, failed to meet this criterion as the percentage of future examples correctly represented decreased as an exponential function of tuple order. Clearly, the 50 training examples they used for all values of N was too few. The questions then asked were: "How many training examples would be required?"; and "How much time is needed to load these examples?" to meet the necessary acquisition of systematicity criterion used in this chapter.

The theoretical result showed that at most a polynomial number of examples were required to attain a high degree of generalization. The empirical results showed that, not only were the number of training examples polynomial, but that the time taken to load these examples was also a polynomial function of N. From these results it was concluded that assuming a restricted architecture (polynomial number of weights) the feedforward network exhibits the necessary acquisition of systematicity, defined in terms of PAC-learnability, with respect to the auto-association of N-tuples task.

It was suggested that the PAC-learnability criterion for the necessary acquisition of systematicity was too weak. It may be the case that people acquire systematic behaviours from fewer examples. In the next chapter, a second criterion, based on linguistic evidence is considered. The feedforward network is then re-examined with respect to this criterion.

Chapter 4

Systematicity as generalization across position

4.1 Introduction

In the previous chapter PAC-learnability was used as the criterion for the necessary acquisition of systematic behaviour against which the feedforward network was examined. This criterion comes from the computer science notion of a *feasible* algorithm and allows one to reject potential cognitive architectures on the basis that they require too much resource (e.g., time) to acquire even modestly sized behaviours.

In this chapter, an alternative criterion called *strong systematicity* (Hadley, 1993) is considered. This criterion, based on linguistic evidence of generalization, is presented in section 2. Feedforward network and recurrent network architectures are examined, by simulation and analysis, on two tasks in which it is possible to exhibit strong systematicity. Those tasks are: the auto-association of 2-tuples, on which it is possible to exhibit strong systematicity of representations (section 3); and querying of 2-tuples (i.e., after presenting an ordered pair, query the network for the first or second component), on which it is possible to exhibit strong systematicity of inference (section 4). A summary of these results and conclusion are given in section 5.

4.2 Hadley's *strong systematicity*

In addition to the work concerned with the generalization capabilities of feed-forward networks in the previous chapter, Connectionists have provided a number of other models that have also demonstrated generalization over structured domains. For example, Elman's (1990) *simple recurrent network* and Pollack's (1990) *recursive auto-associative memory* correctly processed sentences not present in the training set. Hadley (1993) questioned whether the degrees of generalization demonstrated by these and other Connectionist models constitute what Fodor and Pylyshyn intended as systematicity.

Hadley addressed this question by first, reformulating the definition of systematicity based on linguistic evidence; and second, evaluating a number of Connectionist models with respect to this definition.

4.2.1 Strong systematicity defined

Strong systematicity is the above chance level capability of correctly processing sentences containing words in novel syntactic positions. A model or person is said to be strongly systematic if they can, for example, correctly process the sentence *Mary loves John* having never before seen *John* in the *patient* position. In other words, strong systematicity is generalization across syntactic position. (Note that it is not sufficient for a model to correctly process a novel sentence on only one occasion. Say, for example, there are five possible words from which to choose. A network that simply outputs words at random has a 20% chance of correct response. A model's performance must be significantly above this level for it to have exhibited strong systematicity.)

There is ample linguistic evidence of people, and even young children exhibiting at least this degree of systematicity. For example, children are able to coin new verbs from nouns, as in the example provided by Hadley, *It was bandaided*, and subsequently use the new verbs in their passive form. More dramatically, adults given a sentence containing a nonsensical noun (which they are highly unlikely to have seen before) can correctly process sentences with the new noun in other novel syntactic positions.

Strong systematicity is another criterion against which potential cognitive models can be evaluated. In the previous chapter, PAC-learnability was used as the criterion for rejecting potential models on the basis that they require more computational resource than could possibly be available. In this chapter, strong systematicity is used as the criterion for rejecting potential models on the basis that they require more resource (i.e., training examples) than is necessary for people to acquire systematic behaviour.

Given Hadley's strong systematicity criterion then, the question of concern in this chapter is whether Connectionist models, including the feedforward network architecture of the previous chapter, can exhibit this degree of systematicity.

4.2.2 A review of Connectionist models

The starting point for this discussion is with Hadley's conclusion: After reviewing a number of Connectionist models, Hadley (1993) concluded that *Connectionist models have not exhibited strong systematicity.* His conclusion was based on a closer examination of the training scheme applied to the models of McClelland and Kawamoto (1986), Chalmers (1990a), Elman (1989, 1990, 1991), Niklasson and Sharkey (1992), Pollack (1990), Smolensky (1990), and St. John and McClelland (1990). These models, which have all demonstrated generalization over structured domains, could be claimed as refutations of Fodor and Pylyshyn's thesis of compositionality and systematicity without the need for symbol structures. Yet, in each one of these models, there has not been a clear demonstration of strong systematicity. Essentially, a statistical analysis of the training sets revealed that in all probability every word appeared in every one of its allowable syntactic positions in the training set.

Hadley termed the degree of generalization exhibited by these models as being either *weak* or *quasi-systematicity.* Weak systematicity is characterized as generalization to novel sentences in which every word has previously appeared in that syntactic position in the training set. For example, a weakly systematic model is one that could only represent and process the sentence *John loves Mary* having already been trained on sentences where *John* appeared in the agent position, *Mary* appeared in the patient position, and *loves* in the action position, although not necessarily in that combination. *Mary*, for example, could have appeared in the sentence *Tim loves Mary*.

Quasi-systematicity is characterized as generalization to novel sentences containing *embedded* clauses in which every word in the embedded or surrounding clause can be found in the same position of some *simple* sentence in the training set. For example, a model that could only represent and process the sentence *Tim knew John loves Mary* having already been trained on the sentences *Tim knew Sue* and *John loves Mary* demonstrates quasi-systematicity, not strong systematicity, as the words *Tim* and *knew* appearing in the agent and action positions (respectively) in the surrounding clause appeared in the same positions in the simple sentence *Tim knew Sue* in the training set. Similarly, the words *John*, *loves* and *Mary* appearing in the agent, action and patient positions (respectively) relative to the embedded clause, also appeared in the same positions in the training set.

Hadley found that the models of McClelland and Kawamoto (1986), Chalmers (1990a), Elman (1989, 1990, 1991), Smolensky (1990) and St. John and McClelland (1990) have not demonstrated anything more than weak systematicity, and that Pollack (1990), and Niklasson and Sharkey (1992) have not demonstrated anything more than quasi-systematicity. Furthermore, the demonstrations of quasi-systematicity make an unrealistic assumption by preprocessing the input into tree structures.

Subsequent work by Niklasson (1993) claims to have demonstrated strong systematicity on a transformation of logical expressions task. However, component representations presupposed a similarity based on their position, or role within a complex expression. For example, the proposition symbols (p, q, r, s) all shared a common feature in their vector representation. Furthermore, the so-called *novel* component, s, on which the network was tested had as its representation the common feature with no other features (vector components) active. Consequently, the generalization the network demonstrated was due to the common feature on which the network had already been trained in the components of p, q, r, and so cannot be considered a demonstration of strong systematicity. See also Hadley (1993) for a similar comment on McClelland and Kawamoto (1986).

Component representations

The use of similar component representations also occurs in Chalmers (1990a), where words of a common class (e.g., verbs, nouns) share a common feature in their vector representations; and, in Niklasson and van Gelder (1994), where component representations of objects belonging to a common category (e.g., proposition, conjunctive) are constructed with vectors representing these categories. However, this information is, in general, not explicitly available within the external representations of component objects[1]. The information identifying which category a component belongs to is available implicitly in terms of the words relative position in a sentence. These two models assume a similarity that Connectionist networks are expected to learn. Such an assumption begs the question that the criterion of strong systematicity poses: Can Connectionist models acquire position-based similarities and consequently demonstrate generalization across position given category information implicit within the relative positioning of component objects?

The point is that systematicity is a property at the level of compositional objects, not at the level of external component objects. In language, systematicity is a property at the level of sentences, not at the level of words (Fodor & Pylyshyn, 1988). That is, the ability to represent and process sentences of a particular structure relates to other sentences conforming to the same structure.

Orthogonal external representations of components To appreciate the

[1]In the sense that there is some feature of the component object that identifies its category. Information can also come in the form of semantics. Knowledge (past experience) with a word can identify its category. For example, experience with the word *Mary* and its associated object provides one with the information that this word is a noun. However, semantics alone does not necessarily identify a category. For example, *dog* is both a noun and a verb (to chase relentlessly). Furthermore, novel words in isolation have no semantic content.

significance of component representations, suppose a network is required to learn to auto-associate 2-tuples. Suppose, also, that this network is composed of two unconnected subnetworks, where each subnetwork consists of a single input unit connected to a single hidden unit, which is connected to a single output unit. Each component is represented, externally to the network, by a single scalar value between zero and one. For example,

- *Ann* - 0.1;
- *Bill* - 0.3;
- *John* - 0.5;
- *Karen* - 0.7;
- *Vivian* - 0.9.

Auto-associating the tuple (*John*, *Bill*) requires performing the mapping $(0.5, 0.3) \rightarrow (0.5, 0.3)$. Since there is a linear dependency built into the input/output representations, having only seen two of the atomic items, the network would generalize to the others, therefore demonstrating generalization across position. Yet, the two subnetworks are independent of each other. Since they are not connected no information can be conveyed. Thus strong systematicity in this case is a consequence of similarity at the component level.

Of course, there may be *a priori* similarities between component objects. For example, in the set of possible phonemic components of English words, "dee" is closer to "tee" than the "a" sound in "cat". It is an empirical question as to the degree of similarity there is between component objects and its impact of exhibiting systematicity. However, given that systematicity is such a pervasive property across languages and modalities, it is assumed that *a priori* similarity at the component level does not play a significant role in systematicity, which is a property at the level of complex objects. Consequently, when examining Connectionist models for systematicity there must be no *a priori* similarity between the external representations of component objects. In subsequent simulations and analyses, orthogonal (actually local[2]) representations are used.

4.2.3 Summary of review

In all cases, with the exception of Niklasson (1993) and Niklasson and van Gelder (1994), Connectionist modelers did not set out to demonstrate strong systematicity *per se*. Their main concern was to demonstrate some degree of generalization over a structured domain. With that point in mind it is perhaps not surprising that none of the models demonstrated strong systematicity.

However, a counterpoint to this argument is that it seems a remarkable coincidence that, since it is generalization that all modelers are seeking, all models have not shown a greater degree of generalization. This coincidence raises the question of whether there is not some deeper, more fundamental reason why Connectionist models have not demonstrated strong systematicity. The next two sections are directed at determining architectural properties that permit/prevent Connectionist models from exhibiting strong systematicity.

4.3 Strong systematicity of representation

The purpose of this section is to elucidate the architectural properties of particular neural network models that permit or prevent the model from displaying strong systematicity of representation. Before analyzing particular models there first must be a task on which it is possible to exhibit strong systematicity.

4.3.1 Task: Auto-association of 2-tuples

In the previous chapter, auto-association of N-tuples was the task used to evaluate the systematicity of representation of feedforward networks. In language, simple sentences representing binary relations of the form *Noun transitive-verb noun* are instances of 3-tuples. For example, the simple sentence *Mary loves John* is an instance of the 3-tuple (Mary,loves,John). In the case of binary relations, a model

[2]One vector component *on*, the rest *off*.

exhibits strong systematicity when, for example, having only ever been given examples of tuples with *Mary* in the first position, the model generalizes to examples with *Mary* in the third position. With respect to exhibiting strong systematicity over any one binary relator (e.g., loves), the second position component is common to both the trained and generalized examples, and so carries no information regarding generalization. Therefore, assuming that the common component does not incur a significant learning overhead, the domain can be simplified to a set of 2-tuples, where the agent component is in the first position and patient component is in the second position.

The advantage of making this assumption is that it aids in analysis of Connectionist models. Furthermore, a model that cannot demonstrate strong systematicity on this task, will not demonstrate strong systematicity when the common component is included, since, as already mentioned, the common component carries no additional information.

As in the previous chapter, systematicity of representation can be captured by a network's ability to auto-associate the input. In the domain of 2-tuples, ordered pairs are constructed from a set of five atomic objects $\{Ann, Bill, John, Karen, Vivian\}$. A network or model is considered to have demonstrated strong systematicity of representation when it can correctly auto-associate (above chance level on repeated trials), for example, (*Mary*, *John*) having only ever seen *Mary* in the patient (second) position.

4.3.2 Feedforward network

The feedforward network used to examine generalization over a structured domain in (Brousse & Smolensky, 1989; Brousse, 1991) and in the previous chapter had at the input and output layers one unit per component per position. For the task described in this section, this means that there are 10 input units completely connected to a number of hidden units that are, in turn, completely connected to 10 output units (Figure 4.1).

Furthermore, to demonstrate strong systematicity there must be at least one

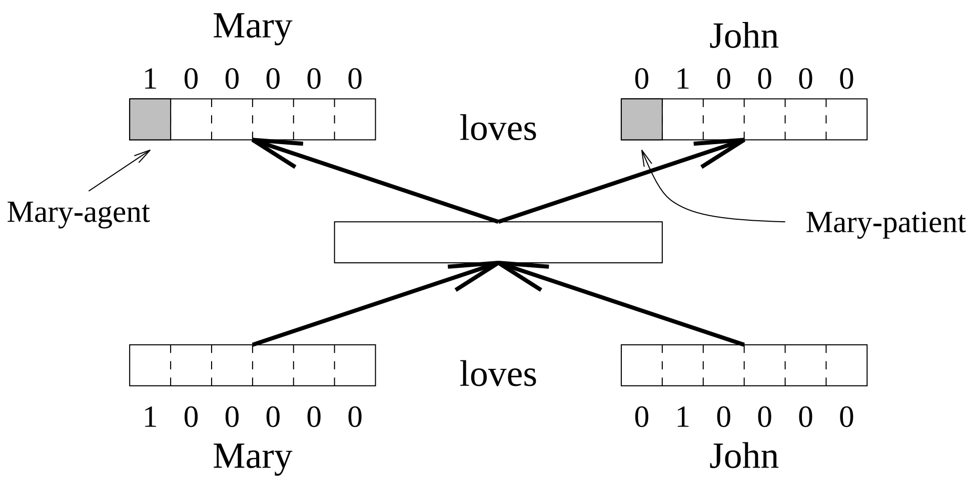


Figure 4.1: The feedforward network architecture for auto-association of 2-tuples. Dashed lines identify individual units within a group of units. The left and right shaded boxes indicate the nodes that extract, from internal representations, the *Mary* component in the agent and patient positions, respectively.

component that does not appear in one of the positions in the training set. Suppose, without loss of generality, that the component *Mary* appears only in the agent position in the training set. The network is considered to have demonstrated strong systematicity if on testing it correctly auto-associated tuples with *Mary* in the patient position.

Independent weights

With this architecture, each output unit is dedicated to detecting a particular component in a particular position from the network's hidden unit (internal) representation of the complex object. For example, one output unit detects the presence or absence of *Mary* in the agent position. The correct performance of this task depends on orientating the *Mary-agent* hyperplane so as to position all points in the training set into two classes: *Mary* in the agent position; and, *Mary* not in the agent position (Figure 4.2). If there are *enough* training points then the network will also correctly classify test cases.

However, for the output unit that detects *Mary* in the patient position there is only one class of points in the training set (i.e., *Mary* not in the patient position), by the requirement of demonstrating strong systematicity (Figure 4.3). With only one

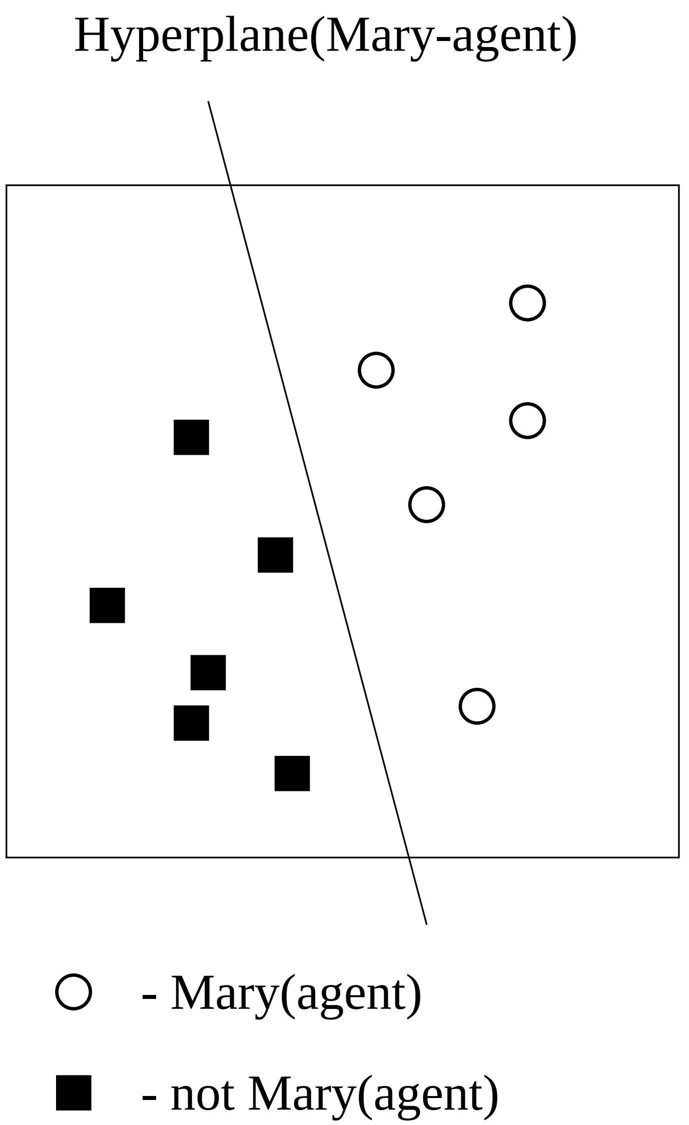


Figure 4.2: Orientation of the Mary-agent hyperplane in hidden unit activation (internal representation) space. The empty circles are points (or vectors) representing structured objects that have as one of its components *Mary* in the agent role. The solid squares are points representing structured objects that do not have as one of its components *Mary* in the agent role.

class of points, the training set provides no information regarding the orientation of the hyperplane to correctly classify points with *Mary* in the patient position, and so the network cannot be expected to generalize from components appearing in the agent position to components appearing in the patient position. That is, the network cannot demonstrate strong systematicity.

In any one trial, it is possible that in the event of correctly classifying examples from the training set, the *Mary-patient* hyperplane is fortuitously positioned so that the network also correctly classifies the test examples where *Mary* appears in the patient position. However, considering the information required to correctly position this hyperplane, this event is highly unlikely. Since the training set contains only one class of point with respect to the hyperplane, there are two choices per dimension of the representation space for correct classification of all points in the training set (i.e., position the hyperplane along the dimension so that all points are to the left or the right of the hyperplane). In an N dimensional space, an estimation of the likelihood of correctly positioning the hyperplane so as to correctly classify the test examples is: $\frac{1}{2^N}$. For the positioning of hyperplanes in the hidden units space, N is the number of hidden units. Thus, the likelihood of generalization across position in any one trial is an exponential function of N that decreases asymptotically to zero. Since the likelihood of generalization across position is below chance level, this feedforward network architecture cannot exhibit strong systematicity.

A similar argument can be made for the input-to-hidden weights. Again, with respect to the input space, the hyperplanes associated with the hidden units are trained on only one type of point along the *Mary-patient* input dimension (i.e., *Mary* not in the patient position). Therefore, along this dimension (which is the only dimension relevant to the encoding of the *Mary-patient* component) there is no information provided in the training set, and therefore the network cannot be expected to demonstrate strong systematicity.

Essentially, the problem with the feedforward network is that there is a separate set of weights that implement the functions that encode/decode the agent and

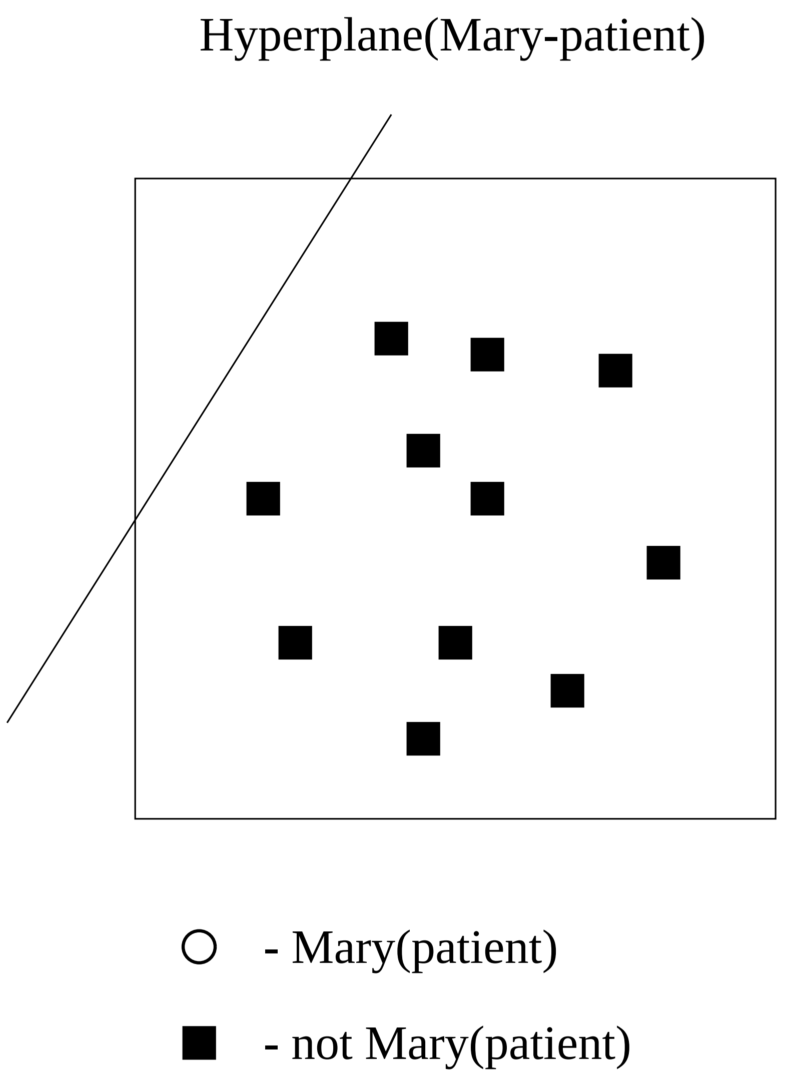


Figure 4.3: Orientation of the Mary-patient hyperplane in hidden unit activation (internal representation) space. The solid squares are points representing structured objects that do not have as one of its components *Mary* in the patient role. There are no points in the hidden unit activation space representing structured objects that have as one of its components *Mary* in the patient position.

patient components (Figure 4.4(a)). Furthermore, the domains of the two input-to-hidden layer functions and the co-domains of the two hidden-to-output layer functions are independent. Consequently, learning these functions necessitates training on every input-output pair from their respective domains and co-domains. The reason these component functions must be trained on every point is that, *a priori*, there is no similarity between the external representations of components.

This result also applies to Smolensky's (1987a) *recirculation* algorithm, which is just a special case of the backpropagation algorithm. The recirculation algorithm applies to a feedforward network where the input to hidden and hidden to output weights are identical. In this case, the three-layer network can be considered as a two-layer network (one input/output layer and one hidden layer). To modify weights, activity at the input/output layer is propagated to the hidden layer (resulting in a vector $\vec{p}_0$), then recirculated back to the input/output layer and then back up to the hidden layer (resulting in a vector $\vec{p}_1$). The weights can be updated as a function of the difference vector $\vec{p}_0 - \vec{p}_1$ using the *delta rule*[3] (Rumelhart et al., 1986) Again, as the algorithm is only trained on one type of point (with respect to *Mary*), the training set does not contain any information regarding the correct weight values for generalization across position.

Weight tying

One method of introducing generalization across position is with the use of weight tying. Weight tying links two or more network weights so that the update of one weighted connection automatically updates the weights of the linked connections. Weight tying has been used successfully by LeCun et al. (1989) as a way of introducing translation invariance for optical character recognition.

Similarly, weight tying could be introduced in the auto-association task to demonstrate strong systematicity by linking the weights that implement the first position mapping (i.e., from input to hidden and from hidden to output) to the

[3]The delta rule is backpropagation for two layers of units (i.e., one set of connecting weights). Backpropagation is applicable to arbitrarily many layers of units, and hence is also called the *generalized delta rule*.

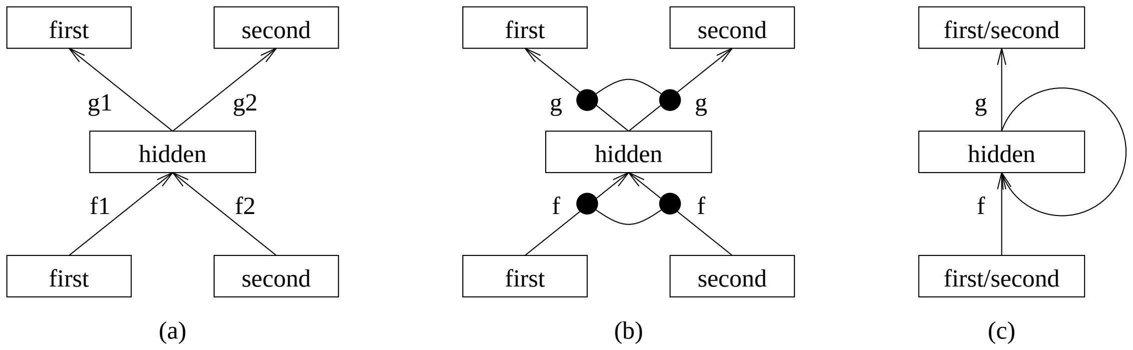


Figure 4.4: Standard feedforward network architecture for auto-association task without weight tying (a) and with weight tying (b). With weight tying, the previously independent functions (f1 and f2, g1 and g2) are now linked by the same set of weights (i.e., f1 = f2 = f; g1 = g2 = g). A recurrent network architecture (c) incorporates partial weight tying by making the input-to-hidden and hidden-to-output layer weights for each component the same weights. In each case, arrows indicate complete connectivity from units in the source layer to units in the destination layer.

weights that implement the second position mapping. Effectively, weight tying makes available multiple copies of the functions that map component representations (Figure 4.4(b)).

The problem with weight tying is that it presupposes knowledge of the very weights that are to be learnt. Furthermore, with complete weight tying it is not possible to extract the two component representations from the internal hidden unit representation because the same hidden to output unit function g is required to perform two different mappings for the same hidden unit vector. There does not exist a set of weights to implement the function g, such that, $g(\vec{h}) = \vec{x}_1$ and $g(\vec{h}) = \vec{x}_2$, where $\vec{h}$ is the internal hidden unit representation, of the 2-tuple (x_1, x_2); and $\vec{x}_1$ and $\vec{x}_2$ are the output representations of the first and second components, respectively.

Recurrency

Another method of incorporating a dependency between component mapping functions is by making use of the same set of input (output) units for presenting (extracting) component representations. In this way, dependency between the component mapping functions is introduced as the functions are implemented over the same set of weights.

Of course, presenting all components over the same set of units means that representations constructed at the hidden layer will be lost on presentation of subsequent components, therefore, preventing the network from constructing complex representations. However, previously constructed representations can be maintained by providing feedback connections that retain representations for future processing (e.g., Figure 4.4(c)). Architectures with feedback connections are called recurrent networks. In the next subsection, a recurrent network is investigated for its ability to demonstrate strong systematicity of representation on the auto-association task.

4.3.3 Simple recurrent network

In the previous subsection, it was shown that the standard feedforward network (i.e., without weight tying) could not demonstrate strong systematicity of representation. Essentially, the lack of strong systematicity was a consequence of the independence of the weights that implement the functions which map components to and from internal representations.

This result suggested a more structured network where components are presented at the same set of input units and consequently, mapped internally by the same set of weights. A recurrent network architecture has such a structure. In general, a recurrent network has a set of input units, a set of hidden units, a set of context units (which hold the hidden unit activations from the previous time step) and a set of output units.

There are a number of ways of connecting each set of units. The architecture used in this subsection is Elman's (1989) *simple recurrent network* (Figure 4.5). The performance of the simple recurrent network on a temporal version of the auto-association of the 2-tuple task is tested and analyzed for its ability to demonstrate strong systematicity.

Simulation

The task of the recurrent network is essentially the same as for the feedforward network described in the previous subsection except that instead of presenting both components simultaneously, components are presented one per time step. After presenting both components, at which time the network should have constructed some internal representation of the ordered pair, the network is required to output the ordered pair in the order that it was presented. Figure 4.5 shows an example of the network and the input-output mapping it is required to perform.

Each trial of the network consisted of:

- Random generation of training examples from a distribution where all five atomic objects can appear in the first position, but only four atomic objects can appear in the second position. The training set was large so that in all likelihood all 20 combinations occurred. Weights were updated at the end of each sequence (i.e., every four patterns). Context units were reset to zero at the beginning of each sequence.

- Random initialization of network weights from a uniform distribution between -1 and 1.

- Training of the network using the standard error backpropagation algorithm with a 0.1 learning rate, no momentum term, and the sum of squares error function (Rumelhart et al., 1986) until performance on the training set was within 0.5 of the target for all output units for all patterns in the training set.

- Testing on all remaining combinations (i.e., every atomic object in the first position combined with the atomic object left out of the second position in the training set).

- Recording of network response to all output patterns in the test set. Two testing criteria were used. A network response was considered correct when: (1) the maximally activated output unit had a target activation of one -

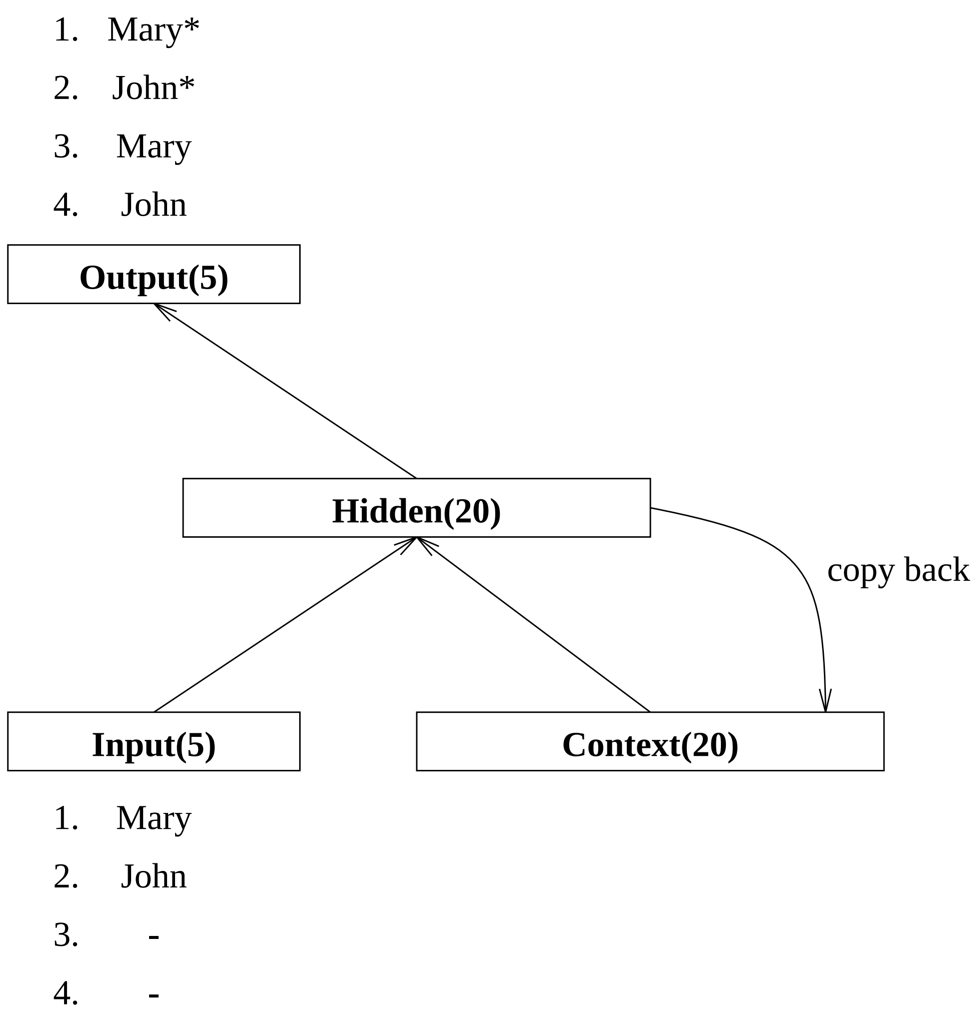


Figure 4.5: The simple recurrent network and the temporal version of the auto-association of 2-tuples task. Parenthesized values indicate number of units used in simulations. Numbers indicate specific time periods. Dashes (-) indicate zero input. Starred (*) output indicates the auto-association of the current input to assist in the formation of internal representations. However, for the purpose of evaluating strong systematicity, only output at time steps 3 and 4 (i.e., auto-association of the ordered pair) was considered.

maximum criterion; or (2) all output units were within 0.5 of their target - 0.5 criterion.

In addition, the number of training examples (ordered pairs) was varied from 10 to 200. Since each network was initialized from a random set of weights, each train and test trial was repeated five times for each training set size.

Result

The recurrent network showed perfect performance (for both maximum and 0.5 testing criteria) on the unseen object in the second position in the test set when trained on 200 ordered pairs. When trained on 50 ordered pairs, performance dropped to a mean of 76% (maximum criterion) and 36% (0.5 criterion) over 5 trials (Figure 4.6).

Discussion

The purpose of this simulation was the demonstration of strong systematicity with respect to the representation of 2-tuples.

Generalization across position occurred in all trials when only one of the items from the second position was left out of the training set and the network was trained on all other combinations[4].

There is, however, a question over the robustness of the demonstration of strong systematicity with this recurrent network. When there were 50 pairs in the training set the network is less likely to have seen all combinations, yet likely to have seen all five objects in the first position and all four objects in the second position. However, generalization to the remaining fifth object in the second position dropped dramatically to 76% (maximum criterion) and 36% (0.5 criterion) averaged over five trials. When only 10 pairs were present in the training set generalization dropped to below chance level performance.

The problem that systematicity poses for Connectionist models is not only the

[4]Since there are only 20 remaining pairs (i.e., 5×4) it is likely that every pair appeared somewhere in a training set of 200.

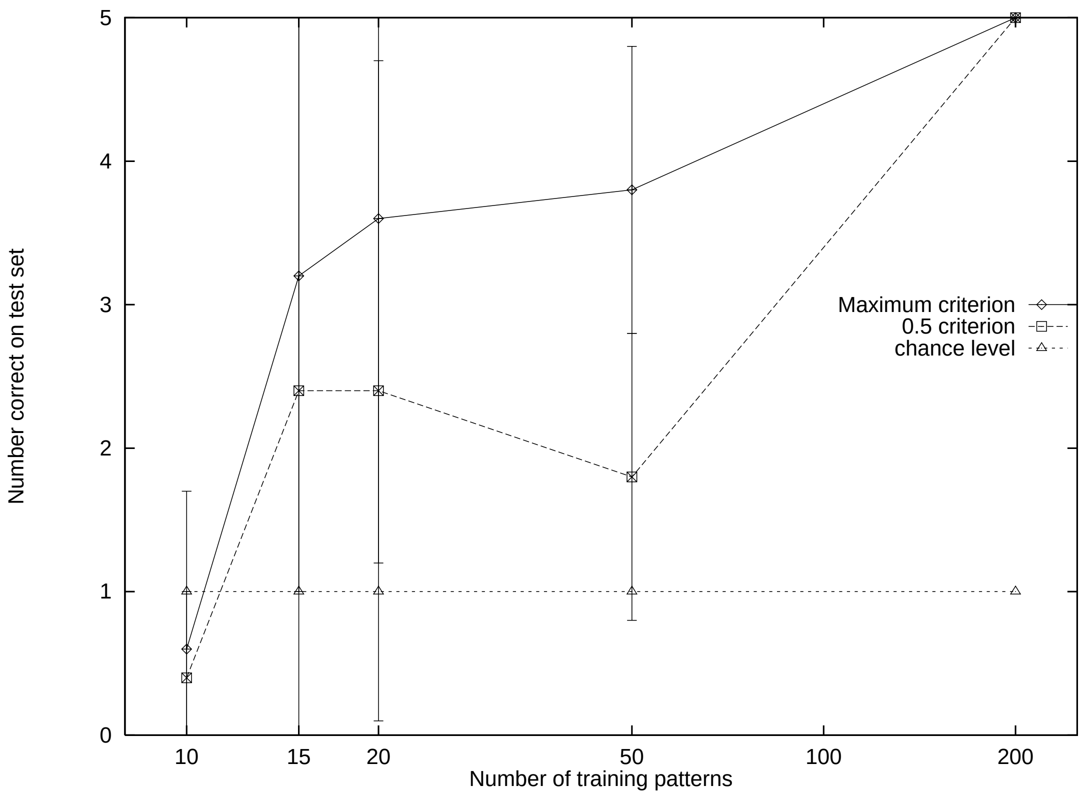


Figure 4.6: Generalization to second position as a function of the number of training patterns. The number of correct responses using maximum (solid line) and 0.5 (dashed line) test response criteria were averaged over 5 trials. Error bars indicate 95% confidence levels. The horizontal dotted line indicates chance level performance. The number of training patterns is plotted on a log scale.

demonstration of such properties as generalization across position, but an explanation as to how such properties necessarily occur as a consequence of the architecture. In other words, regardless of the initial conditions (e.g., initial weight values, specific training sets), how is it that the network consistently demonstrates generalization across position. People, despite their varying biological and environmental backgrounds, demonstrate generalization across position in language. It is this consistency that Fodor and McLaughlin (1990) are referring to when they talk of systematicity being a law of cognitive architecture. Therefore it is important to understand how generalization across position was demonstrated by the simple recurrent network in the case where it occurred in all five trials. In the next subsection, the internal representational structure of the simple recurrent network is analyzed to determine how generalization across position was consistently demonstrated in this case.

Analysis of internal representations

Due to the high dimensionality of a network's internal representational space (which is the number of hidden units) it is, in general, extremely difficult to determine the nature of the internal representations constructed by a network to solve a particular task. However, there are two analysis techniques that can be used to aid an understanding of internal representations. They are: principal components analysis (PCA), first used by Elman (1989) in his analysis of the internal representations constructed by the simple recurrent network on his word prediction task; and canonical discriminants analysis (CDA), first used by Wiles and Bloesch (1992) in their temporal logic task.

A principal components analysis identifies the dimensions of greatest variance of points in some space. The intuition behind its application to networks is that to reduce error the network must learn to separate points corresponding to different target outputs in its hidden unit activation space. Thus, dimensions along which points vary contain important information about the problem, and will be identified by a principal components analysis of points in the hidden unit activation space[5].

Canonical discriminants analysis identifies dimensions in some space that maximizes the ratio of the between-groups variance to the within-groups variance of points in that space. In other words, it finds directions in the space along which points belonging to the same group are tightly clustered, yet separated from points belonging to other groups. This technique is particularly useful in combinatorial domains where one can hypothesize network representations where points are aligned by groups implicit in the training data, but not made explicit in any error signal[6] (Wiles & Ollila, 1993).

The internal representations learnt by the simple recurrent network were analyzed in the case where the network demonstrated generalization across position in all five trials (i.e., when the training set consisted of 200 training examples). After the network was trained to the 0.5 training criterion, all 200 training examples were presented to the network and the resulting 800 hidden unit activation patterns were saved. Principal components and canonical discriminants analysis were performed on these points using the package developed by Dennis and Phillips (1991).

Input to internal subspace The question to be answered is: how are the internal representations organized so as to demonstrate strong systematicity? The first step towards answering this question was to perform a PCA of the hidden unit activations at the first and second time steps (Figure 4.7). The diagram shows the first and second principal components of these points in hidden unit activation space (i.e., the first and second directions of greatest variance, and presumably, greatest information about representational organization). The points are labeled on the basis of the current input, which for the first two time steps are also the target output. The analysis shows that all points can be grouped into five clusters[7]

[5]Thanks to Simon Dennis for this explanation.

[6]In a combinatorial domain, two groups of points could have the same target output which may make it difficult to find using principal components analysis since separation of internal representations is driven, in part, by the error signal. (Of course, separation may also be driven by differences in input representations.)

[7]The cluster labeled "Vivian" only contains one type of point (i.e., *Vivian* at the first time step). Since the network was tested for strong systematicity on this object, *Vivian* did not appear in the second position in the training set.

depending on the output requirement of the network at each point. For example, the input pair (*Bill*, *Ann*) resulted in the point labeled "1Bill"[8] at the first time step, and the point labeled "Bill:Ann" at the second time step.

The activation space is organized in this manner due to the target outputs which at these two time steps were the current inputs. Consequently, error is reduced when points that map to different outputs are separated, and a PCA has identified this separation.

The PCA suggests that the major information contained in the representation is the current object. However, the internal representation at the second time step must still maintain information about the first item, otherwise the network could not possibly recover the first item at time step three. The question is: how is this information maintained?

The PCA of points at the first and second time steps shows some organization within the five clusters. To identify this organization a PCA of points at the second time step only was performed and these points were plotted onto the first and second principal components (Figure 4.8). The plot shows two levels of spatial organization. At one level, points are grouped into four clusters based on their output mapping[9] just as in the previous PCA. At another level, there appears to be the same within-cluster organization of points (labeled by their first object). This consistency is suggested by the same rotational ordering of points within each cluster.

Position independent internal subspaces The consistency of within-cluster organization suggests that first object information is being encoded along some other dimensions independent (by virtue of the apparent regularity) of second object information. If this suggestion is correct then there may exist directions in hidden unit activation space along which all four clusters can be superimposed.

[8]Since the context units were reset to zero at the beginning of each sequence, the hidden unit activations at the first time step were not affected by the previous sequence. Consequently, all sequences with the same first component resulted in the same hidden unit activation vector at the first time step.

[9]Again, there are only four clusters since one object (*Vivian*) was left out of the second position in order to test strong systematicity.

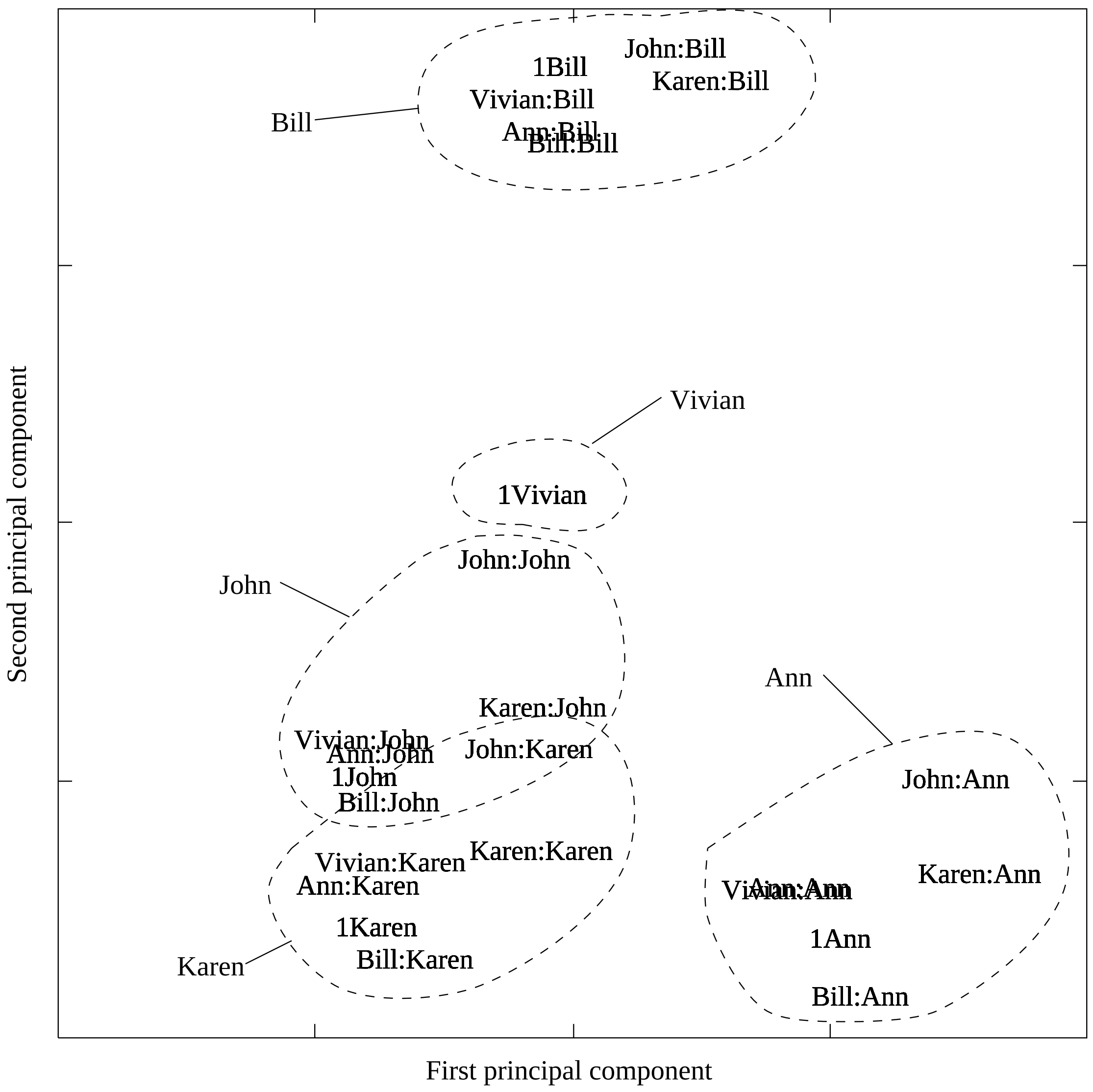


Figure 4.7: Principal components analysis of points in hidden unit activation space generated from training sequences at the first and second time steps. Points are labeled with either the first and second input objects (e.g., Ann:Bill indicates the point as a result of receiving Ann as the first object and Bill as the second object); or, with a single word prefixed by 1 indicating a point as a result of only receiving the first object. Dashed lines group points with the same output mapping.

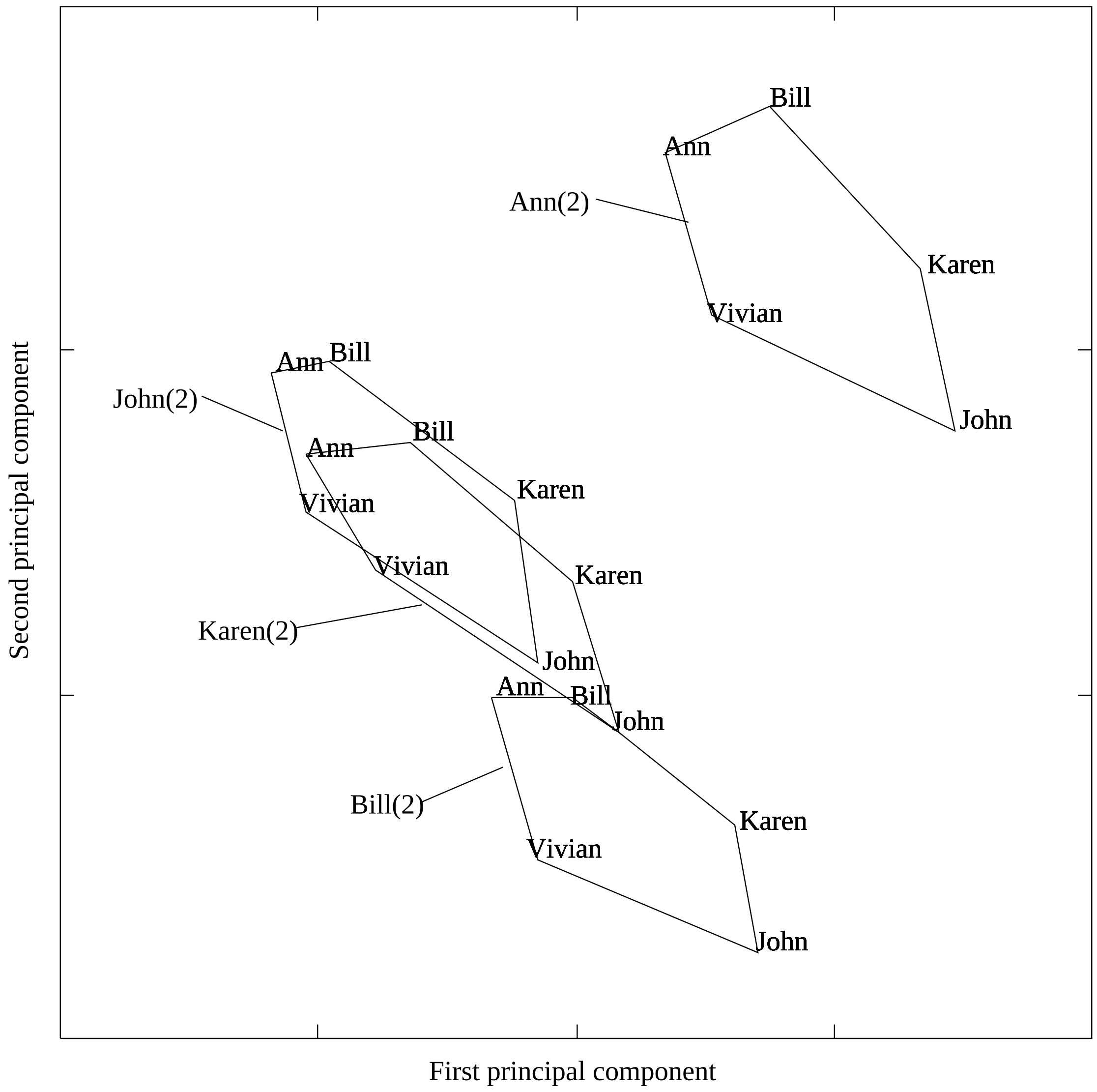


Figure 4.8: Principal components analysis of points in hidden unit activation space generated from training sequences at the second time step. Points are labeled with the first input object. Points that are a result of the same second object are linked by a solid line. Each link is labeled with the object that was presented at the second time step (e.g., points in the top right hand section of the graph are a result of Ann as the second input object).

Or, in other words, there may be a projection from points in hidden unit activation space such that, for example, the point representing the pair (*Karen*, *John*) and the point representing the pair (*Karen*, *Ann*) become the same.

If such a direction exists, it can be found using CDA by grouping all points on the basis of the first object. CDA will identify dimensions in hidden unit activation space such that points in the same group (e.g., (*Karen*, *Ann*), (*Karen*, *Bill*), (*Karen*, *Karen*), and (*Karen*, *John*)) are close together, while points in different groups are far apart.

A CDA was performed on all hidden unit activations at the second time step grouped by the object presented at the first time step. The points were then plotted onto the first and second canonical discriminants (Figure 4.9). The plot shows that the four clusters can be superimposed, and supports the suggestion that first object information was encoded independent of second object information. In each case, all points belonging to the same cluster differed in location by at most 1×10^{-6}. Another way of conceptualizing the representational organization is to say that the network has buffered the input onto independent dimensions.

A CDA was performed on all hidden unit activations constructed at the first and second time steps. The points were grouped on the basis of input component (which was also the target output component) presented at that time step. For example, the hidden unit activations resulting from the object *Karen* having been presented at the first time step or the second time step are identified, for the purposes of CDA, as belonging to the same group. The points were then plotted onto the first and second canonical discriminants (Figure 4.11). The plot shows that the current input object was encoded on the same dimension independent of its position. Thus, suggesting an organization of internal representations of ordered pairs as characterized in Figure 4.10.

Internal subspace to output A PCA of hidden unit activations at time step three was performed and the points were plotted onto the first and second principal components (Figure 4.12). The plot shows two levels of organization similar to time step two.

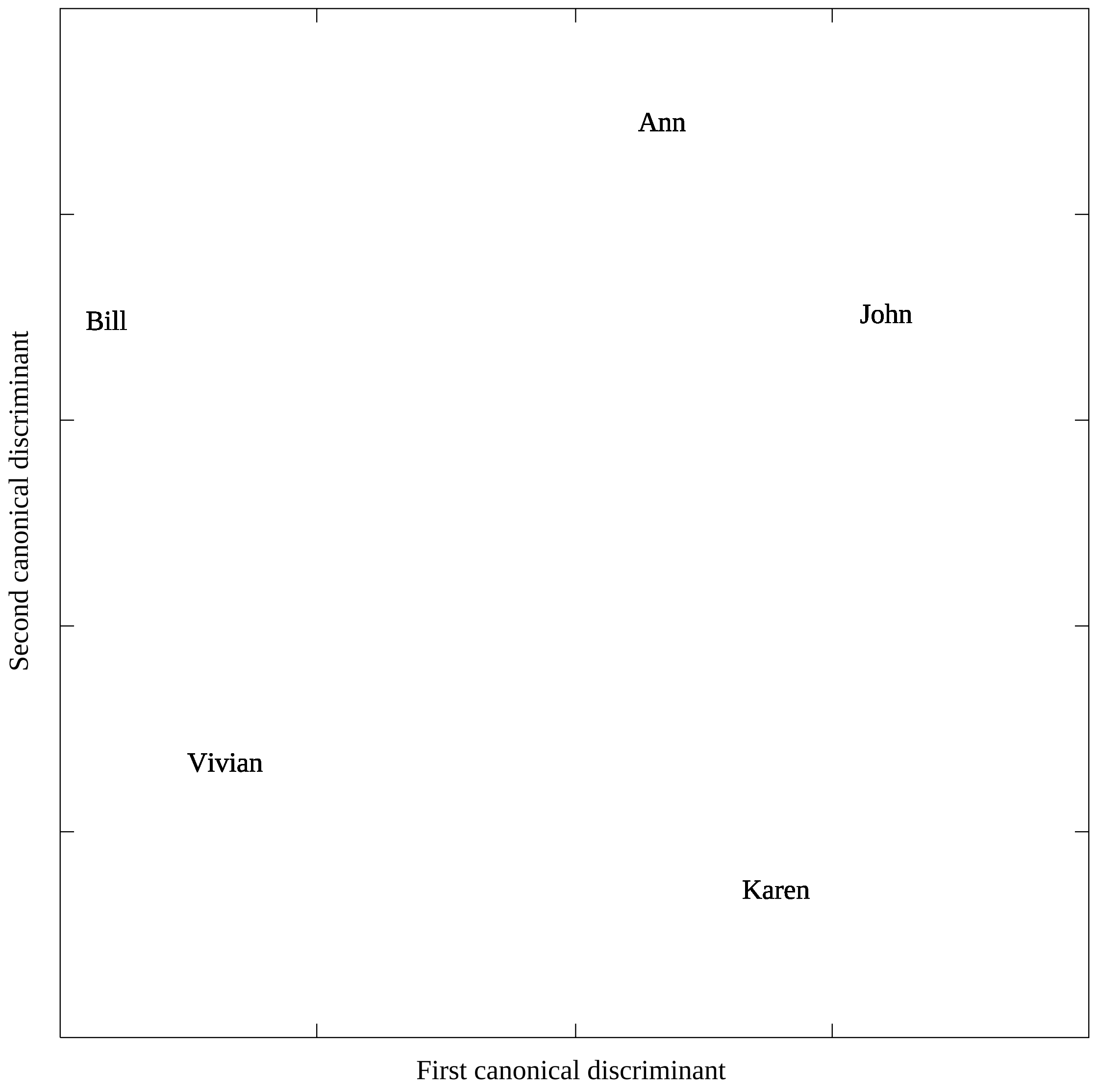


Figure 4.9: Canonical discriminants analysis of points in hidden unit activation space generated from training sequences at the second time step grouped on the basis of the first input object. Points are labeled with the first input object. The fact that there appears only five labels means that all points with the same label have been projected onto the same location.

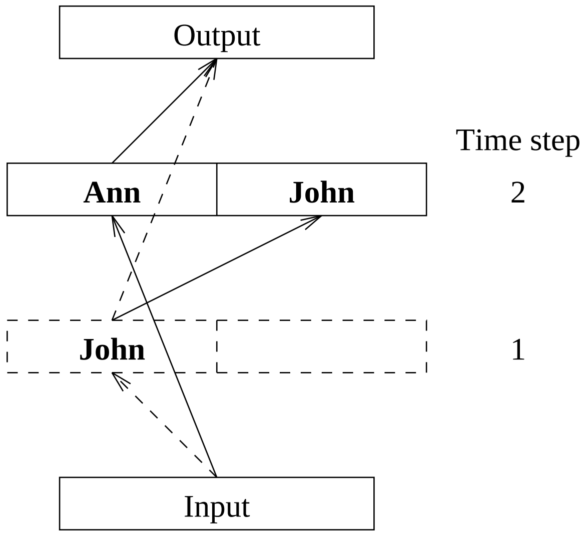


Figure 4.10: Construction of ordered pair representations by buffering input through the same set of dimensions. Dashed lines indicate weights and hidden units at time step one, whereas solid lines indicate the same sets of weights and hidden units but at time step two.

A CDA of these points grouped on the target output at time step three was performed and the points plotted onto the first and second canonical discriminants (Figure 4.13). The plot shows the first object has been maintained on two dimensions independent of the second object.

A CDA of these points grouped on the second input object was also performed. The points were plotted onto the first and second canonical discriminants (Figure 4.14) and shows second object information has been encoded on two further dimensions independent of the first object.

A CDA of points at the third and fourth time steps grouped on the current target output object was performed. The points were plotted onto the first and second canonical discriminants (Figure 4.15). The plot suggests that the current object was extracted along the same dimensions, independent of its position.

First-in first-out buffer This series of analyses suggest that the organizational structure of the simple recurrent network's internal representations is analogous to a *first-in-first-out* buffer (Figure 4.16). By encoding and decoding component representations via the same input and output set of weights (respectively), the network only requires to see each unique object in one of the positions, but not

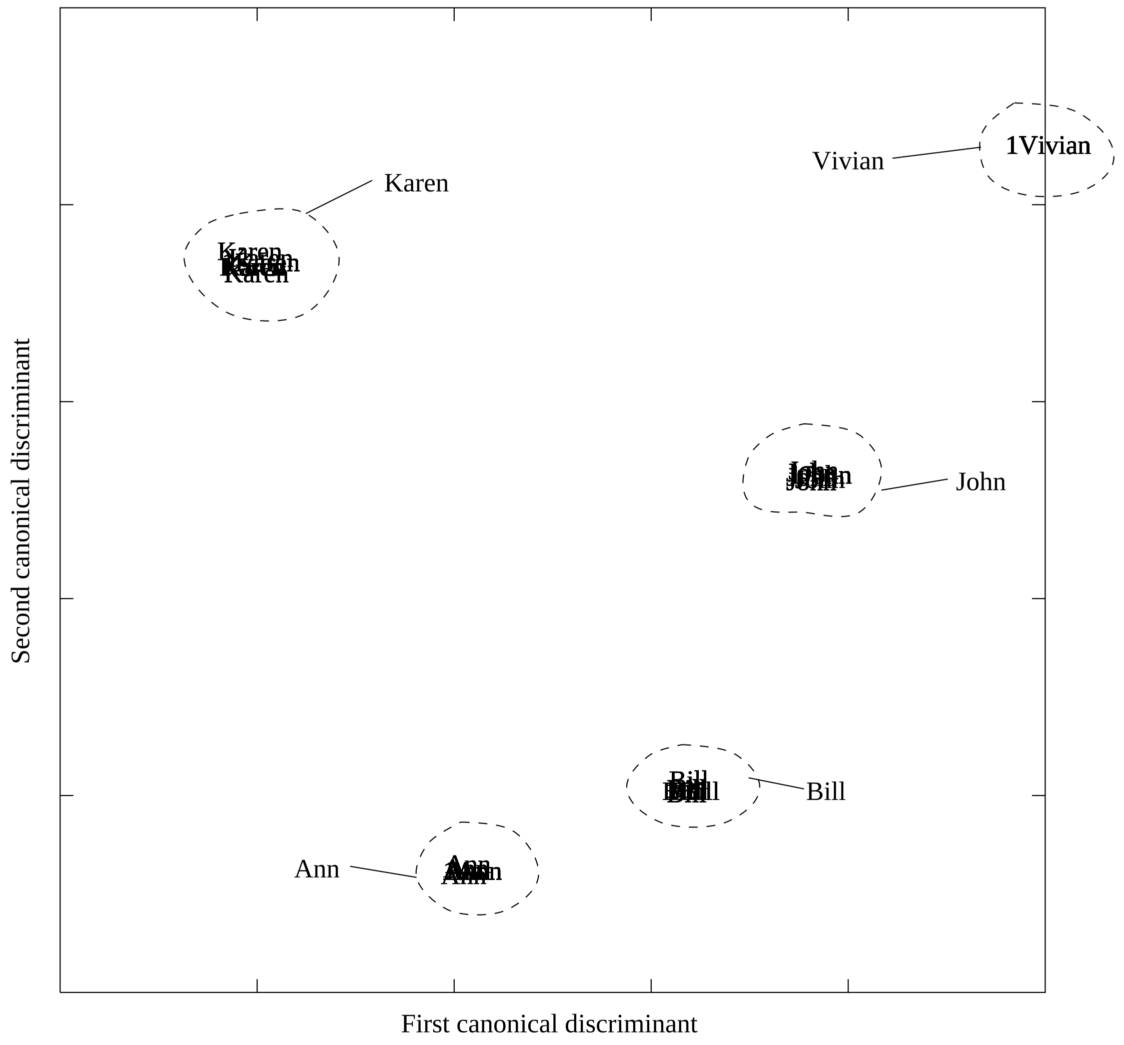


Figure 4.11: Canonical discriminants analysis of points in hidden unit activation space generated from training sequences at the first and second time steps grouped on the basis of the current input/output object. Points are labeled with the input/output object at that time step.

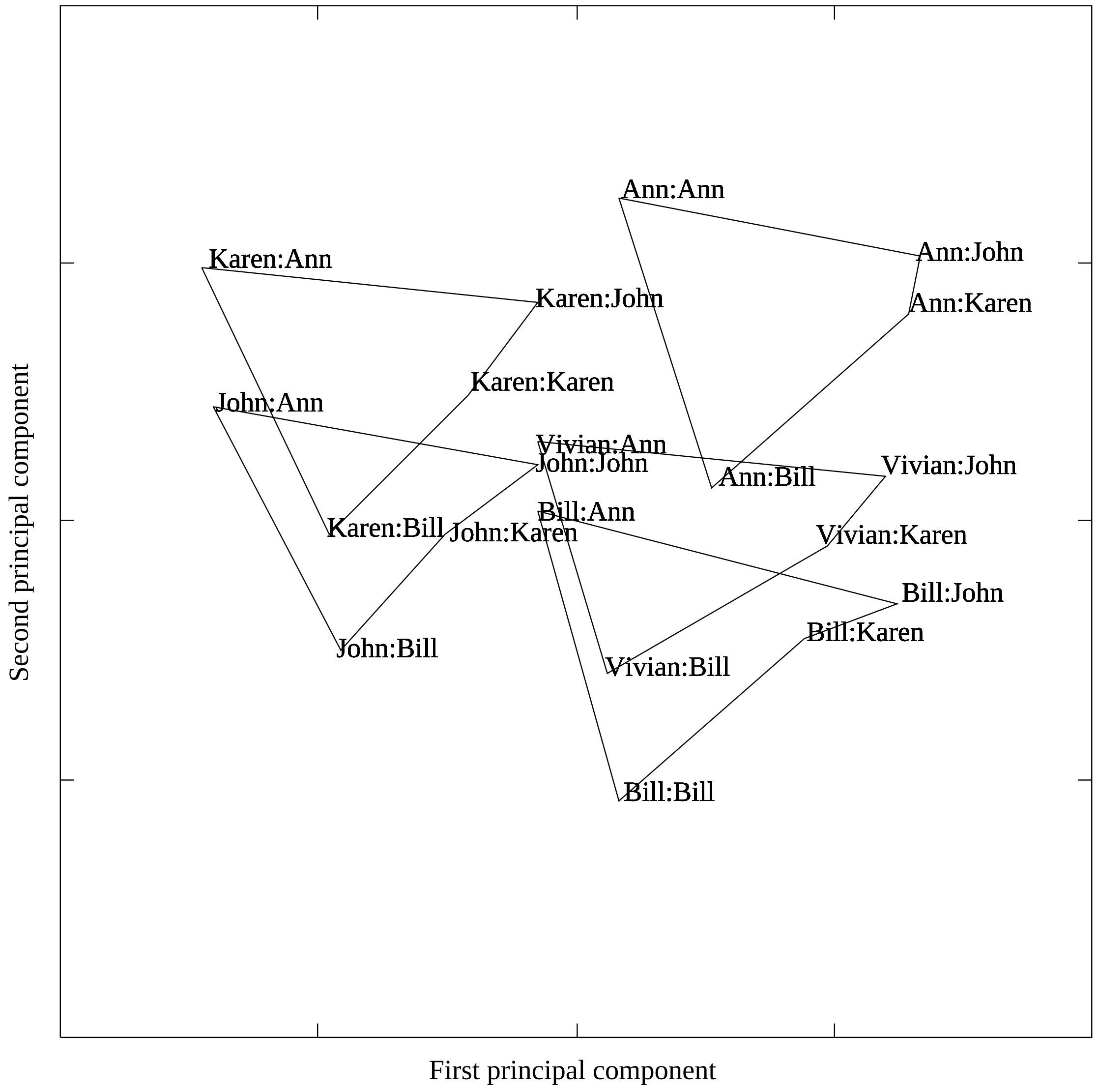


Figure 4.12: Principal components analysis of points in hidden unit activation space generated from training sequences at the third time step. Points are labeled with the first and second input objects (e.g., Ann:Bill indicates the point as a result of receiving Ann as the first object and Bill as the second object). The solid lines link points as a result of the same first input object.

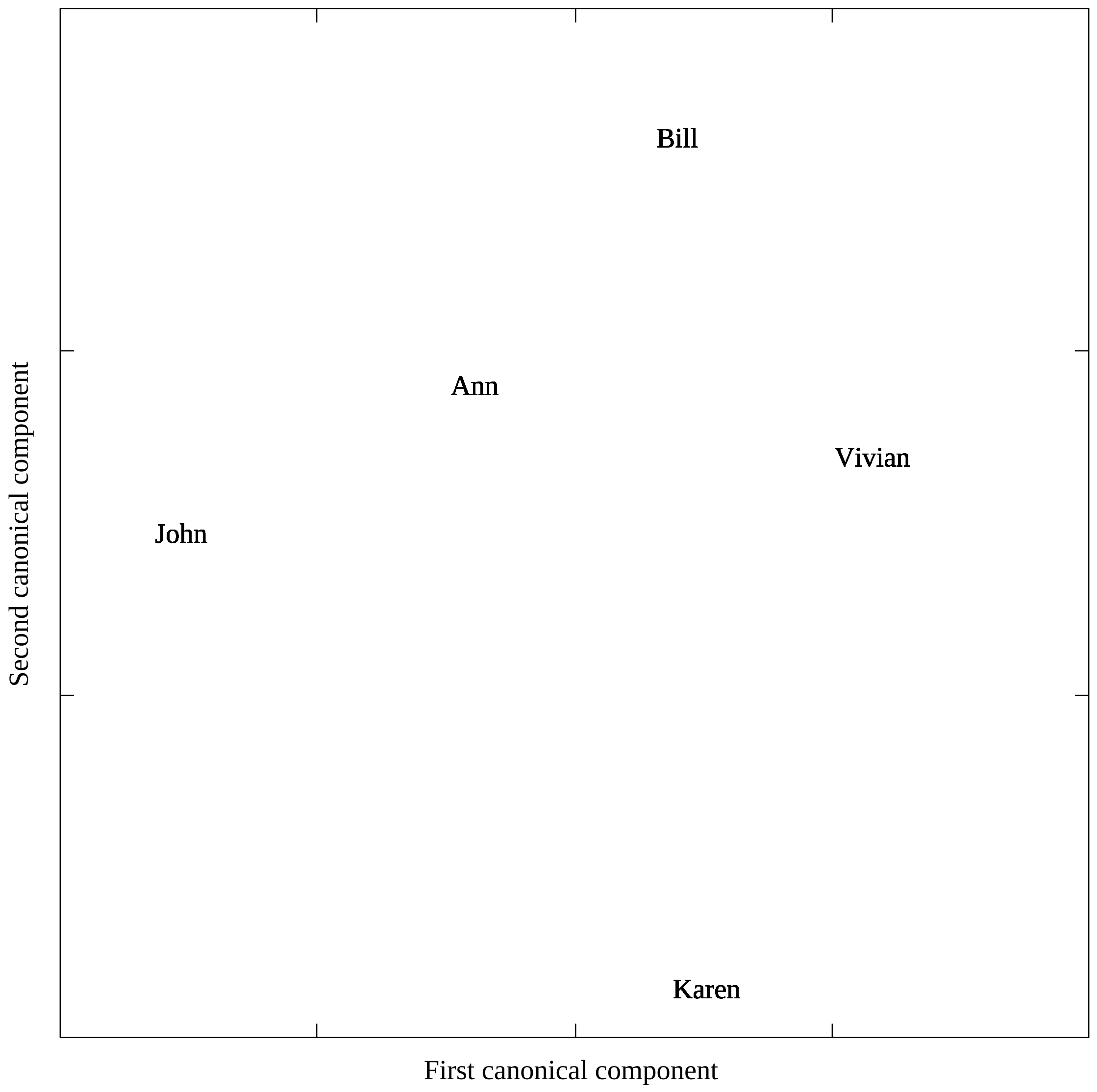


Figure 4.13: Canonical discriminants analysis of points in hidden unit activation space generated from training sequences at the third time step grouped on the basis of the first input object. Points are labeled with the first input object. The fact that there appears only five labels means that all points with the same label have been projected onto the same location.

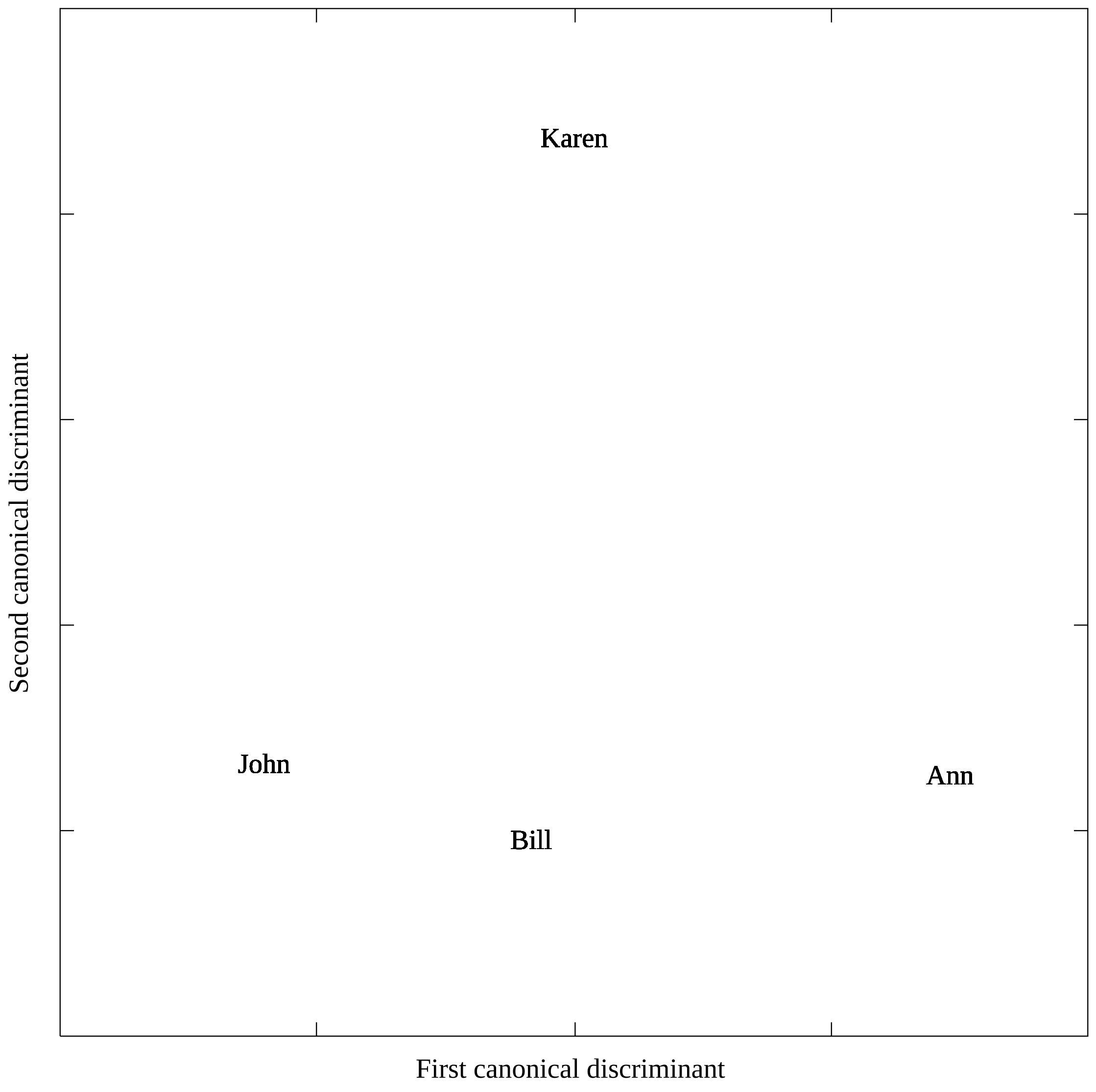


Figure 4.14: Canonical discriminants analysis of points in hidden unit activation space generated from training sequences at the third time step grouped on the basis of the second input object. Points are labeled with the second input object. The fact that there appears only four labels means that all points with the same label have been projected onto the same location.

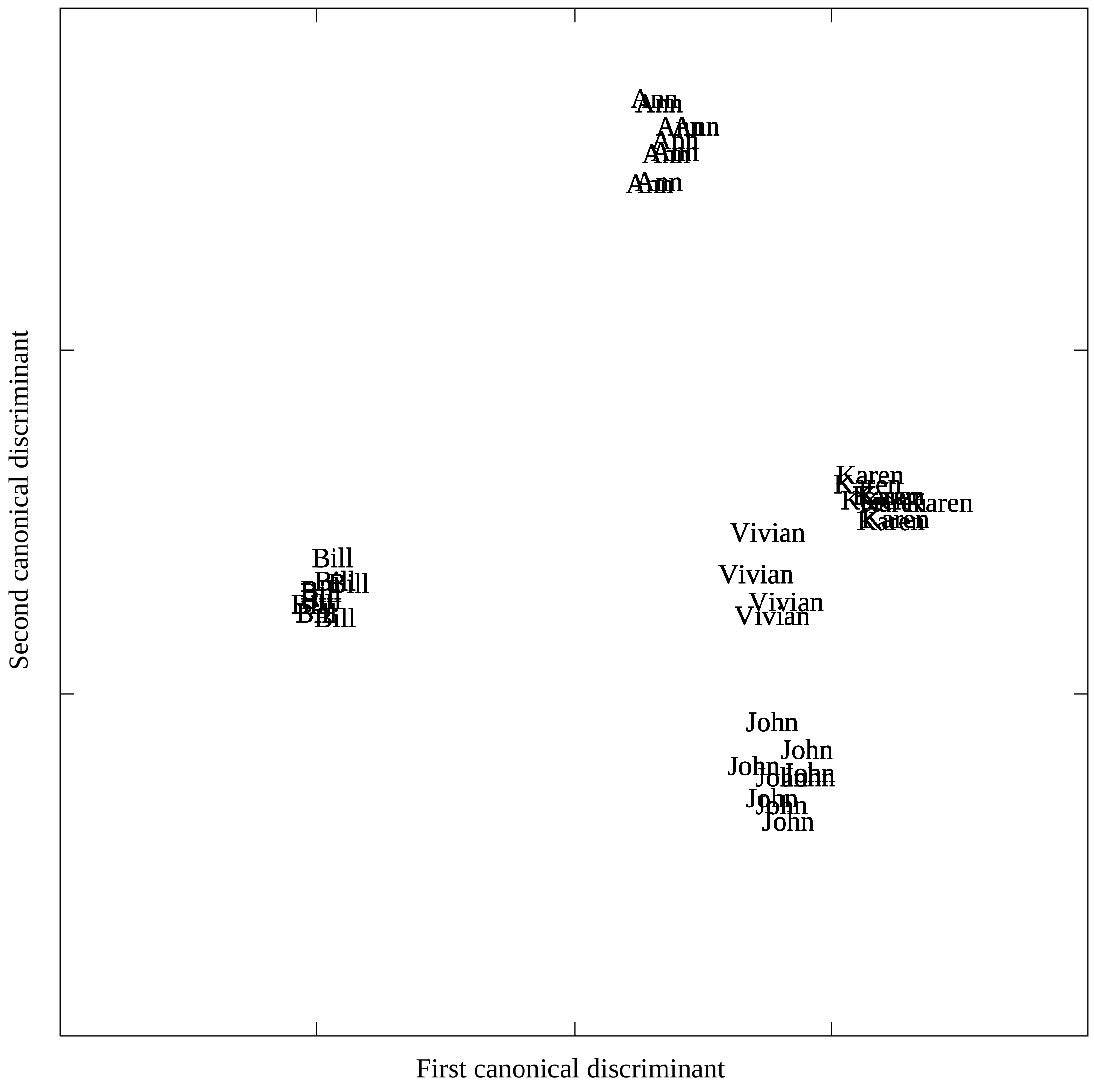


Figure 4.15: Canonical discriminants analysis of points in hidden unit activation space generated from training sequences at the third and fourth time steps grouped on the basis of the current target output object. Points are labeled with the current target output object.

necessarily both, as the simulations demonstrated. When the first component is presented to the network it is mapped to some subspace of the hidden unit activation space. On the next time step, the network maps the internal representation of the first component to a second independent subspace (set of independent dimensions) to make available the first subspace for the second component. At the third time step, the first component is mapped to a third subspace, at which point it is extracted at the output layer, and the second component is mapped to the second subspace. At the fourth time step, the second component is mapped to the third subspace, at which point it is extracted at the output layer. (Note that these subspaces were not localized to specific groups of hidden units. Furthermore, the internal representations were not encoded locally. Figure 4.16 characterizes the structure of hidden unit activation space as being composed of three subspaces. It does not characterize any relationship between subspaces and hidden units.)

The analysis, however, only suggests a possible internal organization for the simple recurrent network, since the two techniques do not consider the hidden to output weights[10]. Nevertheless, PCA and CDA have identified the existance of dimensions along the hidden unit activation space in which component objects were encoded independent of their position. Therefore, learning to extract components along these dimensions will generalize to the other position. Thus, it is not necessary, as the simulation results have shown, for the network to be trained on all components in both positions. It was only necessary that each component appeared in one of the two positions.

Importantly, though, the simple recurrent network only demonstrated perfect generalization in all trials when all other ordered pairs appeared in the training set. When there were fewer ordered pairs in the training set, there were solutions other than the buffer solution which satisfied the requirements of the training set, but which did not demonstrate generalization across position. In these cases, the simple recurrent network required more information than was available in the training set to demonstrate perfect generalization in all trials.

[10]It is, in general, difficult to visualize the relationship between hyperplanes and internal representations for high dimensional spaces.

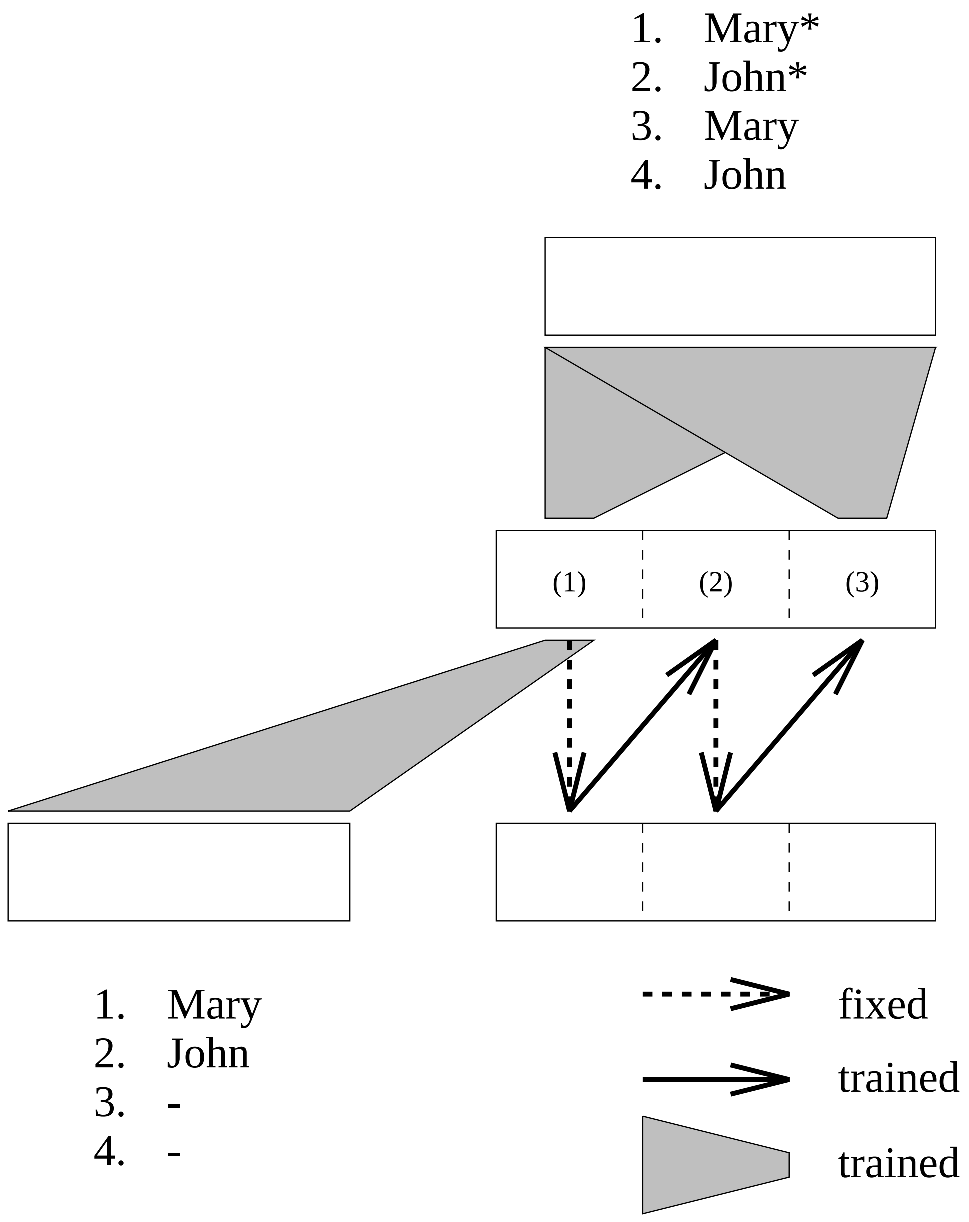


Figure 4.16: Idealization of the *buffer* solution by the recurrent network in demonstrating strong systematicity of representation. Shaded regions and solid arrows indicate modifiable weights. Dashed arrows indicate fixed weights. Starred (*) outputs at time steps 1 and 2 indicate auto-association of current input to assist in the formation of internal representations. Dashed (-) inputs at time steps 3 and 4 indicate zero input. Parenthesized numerals indicate subspaces of the hidden unit activation space.

4.3.4 Summary of strong systematicity of representation

With respect to systematicity of representation it was shown that the standard feedforward network could not demonstrate strong systematicity. The property of the feedforward network that prevents it from exhibiting strong systematicity of representation is the independence of weights that implement component mappings. This independence means that there are too many positions in weight space that implement the function that is specified by the training set. Consequently, there is no information in the training set to determine, with confidence greater than chance level, the function that will exhibit generalization across position on future examples.

This result suggested an architectural bias whereby there is a dependency between the weights that implement component mappings. The simple recurrent network incorporates such a bias. Since component representations are presented to the network at the same set of input units they are mapped, in part, by the same set of weights. Simulations showed that the simple recurrent network could demonstrate strong systematicity of representation with respect to the 2-tuple task. The organization of internal representations learnt by the network was characteristic of a *first-in-first-out buffer*, where components were mapped from the input to a common subspace (input phase), and from a common subspace to the output (output phase). Because these mappings were implemented by the same set of weights it was not necessary for the network to be trained on every component in both positions. It was only necessary that components appear in at least one of the positions. Consequently, the network was able to demonstrate strong systematicity of representations on this task.

4.4 Strong systematicity of inference

Systematicity of inference is the ability to extract components from structurally related objects. For example, one does not find people who are able to infer the *John went to the store* from the statement *John and Mary went to the store*, but cannot infer that *Mary went to the store* from the statement *Mary and John went to the store.* As in the previous section, Hadley's strong systematicity is used as the criterion for determining whether a Connectionist model demonstrates systematicity of inference.

4.4.1 Task: Querying 2-tuples

In this systematicity of inference task, a network is presented with an ordered pair and a query/question, which requests the first or second component of the ordered pair. For example, given the pair (*John*, *Bill*) and the question that requests the first component the network should respond with *John.* As in the auto-association of 2-tuples task, there are five possible atomic objects that may appear in either the first or the second positions. In addition, there are two query objects that request either the first or second component of the ordered pair.

A network is said to have demonstrated strong systematicity of inference with respect to this task if on the test set it can correctly infer components in component-position combinations that did not occur in the training set. In the next subsection, the simple recurrent network is examined for its capacity to exhibit strong systematicity of inference with respect to this task.

4.4.2 Simple recurrent network

The simple recurrent network, which demonstrated strong systematicity of representation, is examined here on this systematicity of inference task.

Simulation

The simulation conditions for the querying of 2-tuples task are as follows:

- Local encoding of input and output vectors. As there are seven possible input vectors in this task (i.e., 5 possible components plus 2 possible queries) there are 7 input units to encode the input vectors, and 5 output units to encode the 5 possible components.

- Generation of 40 training sequences (each sequence consisting of three patterns), consisting of all 5 components in the first position combined with 4 possible components in the second position combined with the two possible queries. The fifth component was used to test strong systematicity. Thus, the test set consisted of 10 sequences (i.e., all 5 components in the first position combined with the fifth component in the second position combined with the two queries).

- Random initialization of weights from a uniform distribution in the range -1 to 1.

- Training of the network using the standard error backpropagation algorithm with 0.1 learning rate, no momentum term, and the sum of squares error function (Rumelhart et al., 1986) until performance on the training set reached criterion. Two training criteria were used: (1) all output units were within 0.5; (2) all outputs were within 0.4 of the target output for every unit on every training pattern. If the network did not reach criterion by the 10000th epoch (where one epoch is the presentation of every training pattern) then training was terminated. Weights were updated at the end of each sequence (i.e., every three patterns). Context units were reset to zero at the beginning of each sequence.

- Testing on all remaining patterns using two criteria for correctness: (1) the maximally activated output unit corresponds to the target activation of 1 - maximum criterion; (2) all output units are with 0.5 of their target activation - 0.5 criterion.

- Each train-test trial was repeated 10 times for each training criterion, with weights being randomly initialized at the beginning of each trial.

The simple recurrent network was examined with 20, 10 and 8 hidden units. In the case of the 8 hidden unit network, 0.5 was used as the training criterion. Figure 4.17 shows the network and an example sequence of input-output pattern pairs.

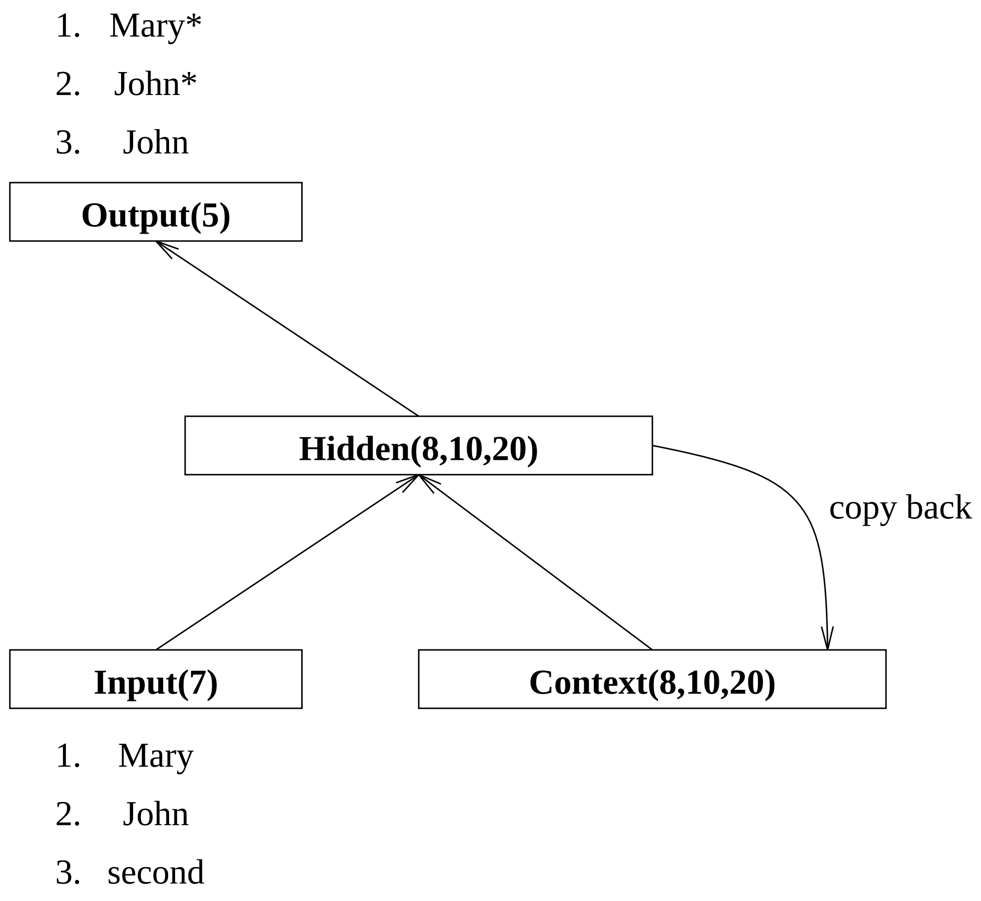


Figure 4.17: The simple recurrent network and the querying of 2-tuples task. Numbers indicate specific time periods. Parenthesized values indicate number of units used in simulations. Starred (*) output indicates the auto-association of current input to assist in the formation of internal representations. However, for the purpose of evaluating strong systematicity, only performance on time step 3 was considered.

Results

In the case of 20 hidden units, the simple recurrent network learnt perfectly on the training set on each of the 10 trials for both 0.5 and 0.4 training criteria (i.e., all output units were within 0.4 of their target outputs for all patterns in the training set). However, in all trials performance on the test set was zero for both maximum and 0.5 testing criteria where the network was required to extract the second component (i.e., the component which did not appear in the second position in the training set). The network did not correctly respond, in any trial, to any of the 5 sequences where the network was required to extract the second component.

In the case where 10 hidden units were used, training reached the 0.4 criterion in 7 of the 10 trials. In all trials, regardless of whether the network reached the training criterion, the performance on the test set was zero, for both testing criteria, with regard to the extraction of the second component.

When only 8 hidden units were used only 1 of the 10 trials converged to the 0.5 training criterion. In all but two trials performance was zero, with 0.5 and maximum testing criteria, on the extraction of the second component in the test set. For the other two trials, test set performance was 1 out of 5 for maximum testing criterion, 0 out of 5 for 0.5 testing criterion. Since there are 5 possible components to choose from this level of performance is no better than chance. Furthermore, in these two trials the network did not acquire perfect performance on the training set within 10000 epochs. The results are summarized in Table 4.1.

Discussion

The purpose of this simulation was to determine whether the simple recurrent network could exhibit strong systematicity of inference. Although simulation results cannot show that the network is incapable of demonstrating strong systematicity on this task, they do suggest some inherent problem with the architecture. As with the systematicity of representation task, the training sets were constructed so as to maximize the possibility of exhibiting strong systematicity by leaving out only one of the five components in the second position and training on all other

Table 4.1: Summary of the performance of the simple recurrent network on the systematicity of inference task for networks with 20, 10 and 8 hidden units. The networks were trained to 0.5 and 0.4 training criteria, or until 10000 epochs of training. In the case of 8 hidden units, only the 0.5 training criterion was considered. The table shows that for a network with 20 hidden units, all 10 trials converged to the 0.5 and 0.4 training criteria within 10000 epochs. For each training criterion, the network was tested on the 0.5 and maximum testing criterion for the 5 test cases. The table shows that for a training criterion of 0.5 and a testing criterion of 0.5, 0 out of 5 test cases were correctly inferred in all 10 trials. In the case of the 0.5 training criteria and maximum testing criterion, in 2 of the 10 trials the network with 8 hidden units correctly inferred 1 of the 5 test cases.

Hidden units	20	10	8*
Convergent trials (0.5/0.4) (10 trials)	10/10	10/7	1
Correct test cases (5)			
Train: 0.4; Test: (0.5/max)	0/0	0/0	-
Train: 0.5; Test: (0.5/max)	0/0	0/0	0/0 (8 trials) 0/1 (2 trials)

combinations. Thus, with respect to this task, the training set provides the maximum amount of information[11] available for a demonstration of generalization across position. Yet, despite being given such information the networks did not demonstrate strong systematicity. Having considered two different training criteria, two different testing criteria, and various numbers of hidden units, in all but two trials, the simple recurrent network performance on the test set was zero for the correct extraction of second position components in the test set. In the other two trials, a network with 8 hidden units, trained to the 0.5 training criterion, and tested on the maximum testing criterion, correctly extracted second position components in only 1 of 5 test cases. In the case of the maximum testing criterion, where output units are randomly activated, there is a 1 in 5 chance (since there are 5 possible components) that the maximally activated output unit will correspond to the target activation of 1. Since performance on these two trials was not greater than chance level performance, the network did not exhibit strong systematicity.

[11] Providing more information in the way of additional training patterns would mean that strong systematicity could not be tested as all components would have appeared in both positions.

The network with 20 hidden units was the same as the network which demonstrated strong systematicity of representation. Although this network did not demonstrate strong systematicity of inference is did perform perfectly on all first queries in the test set. Thus, although not demonstrating generalization across position, the network did demonstrate generalization to novel combinations. The lack of generalization across position suggests that the 20 hidden unit network was not sufficiently constrained in its weights space. Reducing the number of hidden units decreases the region of weight space within which the network can implement the target function. Therefore, given that the network learns the function specified by the training set, there is a greater likelihood of exhibiting strong systematicity. However, decreasing the number of hidden units made learning the training set more difficult to the point where only 1 of the 10 trials terminated within 10000 epochs in the case of the 8 hidden unit network. The error profiles for the 8 hidden unit network in each of the 10 trials are given in Figure 4.18. They suggest that the network was not likely to converge in the near future.

The error profiles show that the error on the training set was generally lowest around the 500th epoch of training. Further training resulted in an increase in training error to a relatively stable point by the 10000th epoch. It is possible that the network at this point could have exhibited generalization to the novel component-position, but at the expense of being unable to correctly infer all cases in the training set.

To test this possibility, the simulations were rerun under the same conditions except that training was terminated by the 500th epoch, at which point the network was tested using the maximum testing criterion.

In none of the 10 trials conducted did the network with 8 hidden units correctly infer the component object in any of the 5 test cases. It would suggest that the network was not generalizing across position, above chance level, at any stage during the 10000 epochs of training in the previous simulations.

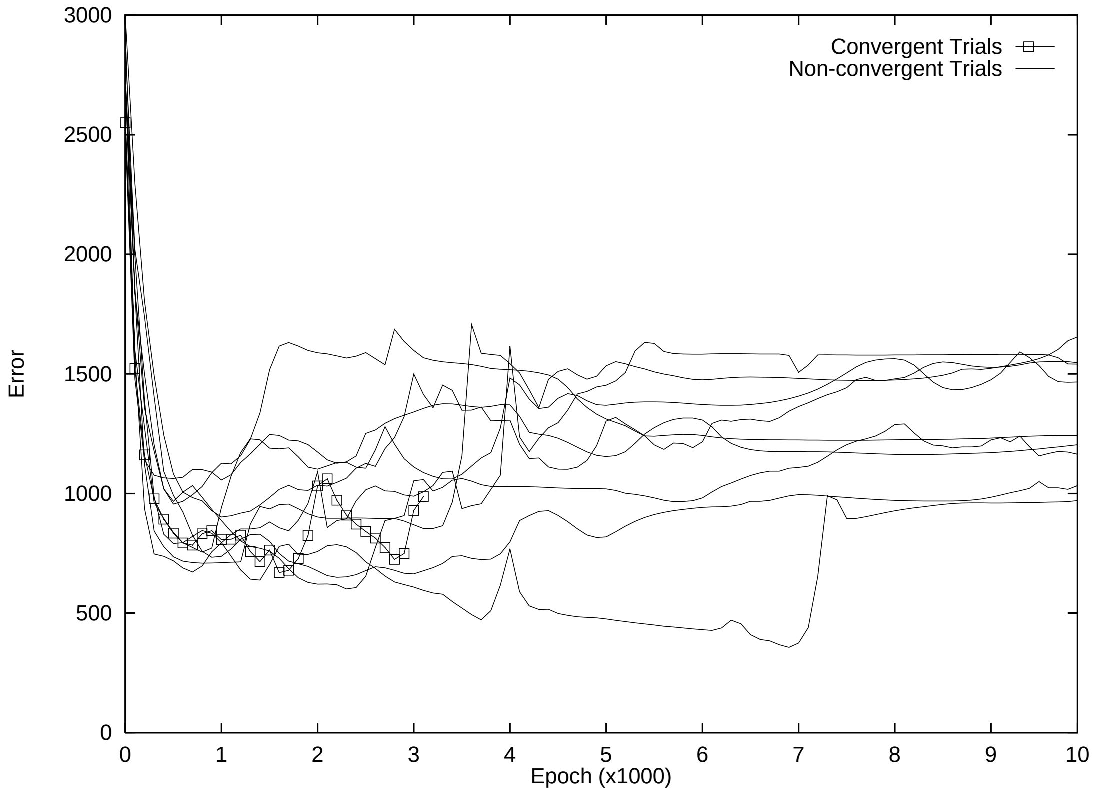


Figure 4.18: Error profile for the 8 hidden unit network over 10000 epochs of training for each trial. Lines indicated trials that did not converge to the 0.5 training criterion within 10000 epochs. The line with points labeled as squares indicates convergence to the 0.5 training criterion within 10000 epochs.

Analysis of internal representations

The lack of strong systematicity of inference from a network that demonstrated strong systematicity of representation raises the question of whether the simple recurrent network is sufficiently biased for this task. In this subsection, by analysis of internal representations it is shown that this simple recurrent network cannot demonstrate strong systematicity of inference with respect to this task. The explanation proceeds by first, showing the minimum network configuration to solve this task; and second, showing that with this minimum configuration the training set does not provide enough information to make generalization across position a likely property.

The simple recurrent network is an instance of a first-order network, characterized by a unit activation function of the form:

$$f_j(\sum_{i=1}^{n} w_{ij}x_i + b_j)$$

where f_j is the activation function for unit j; x_i is the activation from a previous unit i along a connection with weight w_{ij}; and b_j is a bias on the unit. In a first-order network there are no multiplicative activation terms. Consequently, input points resulting in the same unit activation belong to a hyperplane whose orientation is determined by the unit's incoming weights and bias. A hyperplane divides the unit's input space into two half-spaces so that all points in one half-space result in a unit activity (output) greater than some threshold value (θ), and all points in the other half-space result in a unit activity less than θ.

In the querying of 2-tuples task, the network is presented with an ordered pair and a question requesting the first or second component of the pair. Now suppose, without loss of generality, *Mary* is the component on which the network is to demonstrate strong systematicity. Then, to solve the task the network must implement the following function:

$$\begin{aligned}
f(\mathcal{Q}_1, \mathcal{R}[(Mary, X)]) &\rightarrow Mary \\
f(\mathcal{Q}_1, \mathcal{R}[(X, Mary)]) &\rightarrow \neg Mary \\
f(\mathcal{Q}_2, \mathcal{R}[(Mary, X)]) &\rightarrow \neg Mary
\end{aligned}$$

$$f(\mathcal{Q}_2, \mathcal{R}[(X, Mary)]) \;\rightarrow\; Mary$$

where $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the question vectors; $\mathcal{R}[(Mary, X)]$ is a representation of an ordered pair with *Mary* in the first position and some other object (X) in the second position; $\mathcal{R}[(X, Mary)]$ is a representation of an ordered pair with *Mary* in the second position and some other object (X) in the first position; and, f is the function that implements the mapping. The $R[.]$ notation is used to refer to the network's internal representation of an object which may be different from the objects external representation.

A first-order network that implements the mapping is given in Figure 4.19. For a simple recurrent network the representation of each pair at the context units is a result of cycling back representations of components at the hidden units from previous time step. For a feedforward network the representations at the context units can be regarded as just input representations, and the context units can be regarded as just other input units.

As in the previous section, by using a local encoding for each component the assumption is made that there is no *a priori* similarity between components. Therefore, at the output layer there is a single unit that discriminates on the basis of whether or not a representation contains the *Mary* component. That is, there is an output unit that detects *Mary*. It is straightforward to show that this discrimination cannot be performed by positioning a single hyperplane.

To correctly discriminate the *Mary* component a hyperplane must be partitioned so as to satisfy the following inequalities:

$$\vec{W}_I\, \vec{\mathcal{Q}}_1 + \vec{W}_C\, \vec{\mathcal{R}}^C_{Mary,X} + B > \theta \tag{4.1}$$

$$\vec{W}_I\, \vec{\mathcal{Q}}_1 + \vec{W}_C\, \vec{\mathcal{R}}^C_{X,Mary} + B < \theta \tag{4.2}$$

$$\vec{W}_I\, \vec{\mathcal{Q}}_2 + \vec{W}_C\, \vec{\mathcal{R}}^C_{Mary,X} + B < \theta \tag{4.3}$$

$$\vec{W}_I\, \vec{\mathcal{Q}}_2 + \vec{W}_C\, \vec{\mathcal{R}}^C_{X,Mary} + B > \theta \tag{4.4}$$

where $\vec{W}_I$ and $\vec{W}_C$ are the input to hidden and context to hidden weight vectors (respectively); B is the bias (weight) to the unit; and, θ is the threshold above

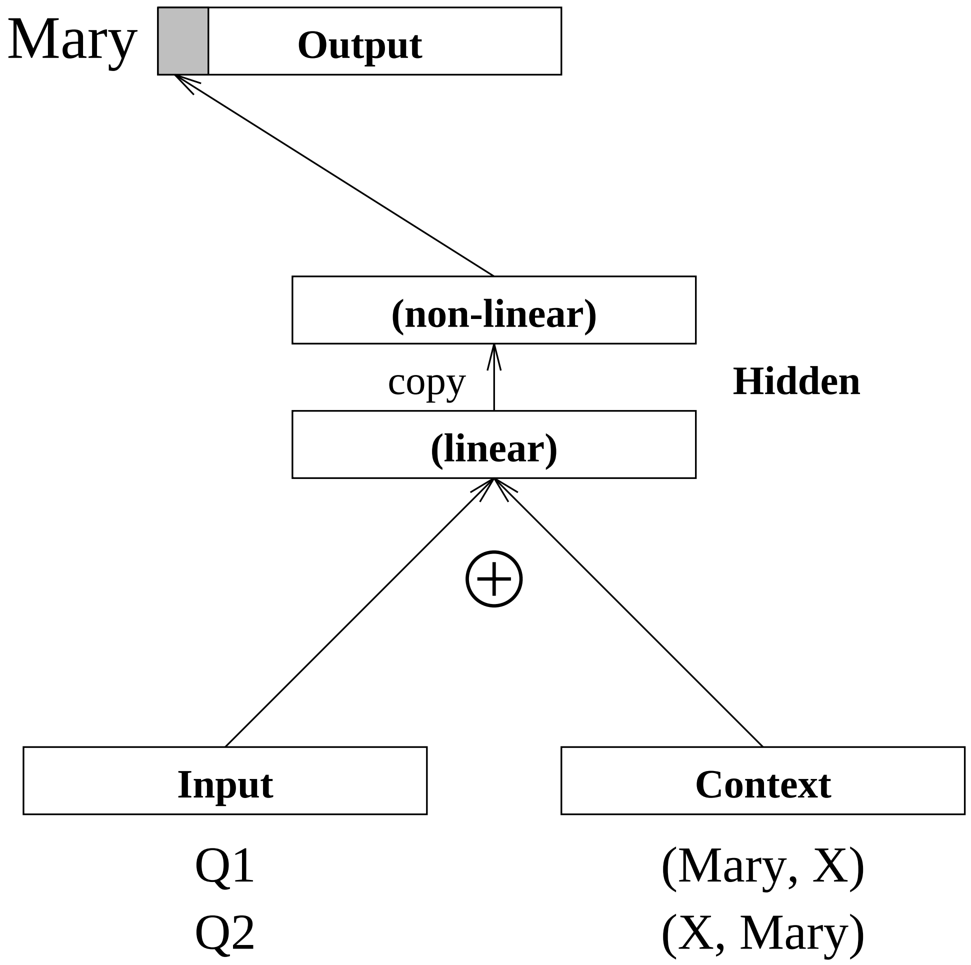


Figure 4.19: Conceptualization of a first-order network where the hidden layer is composed of two layers: the first performs a linear transformation of activation from the preceding layer; and, the second performs a non-linear transformation of the linear layer.

which the component *Mary* is considered to be the correct response.

Subtracting equation 4.3 from equation 4.1, and equation 4.4 from equation 4.2 leaves:

$$W_I\,(\vec{Q}_1 - \vec{Q}_2) > 0 \tag{4.5}$$

$$W_I\,(\vec{Q}_2 - \vec{Q}_1) > 0 \tag{4.6}$$

Since there does not exist a weight (W_I) vector that satisfies both inequalities 4.5 and 4.6, the network cannot represent the solution with a single hyperplane for each component. Consequently, to correctly extract the *Mary* component requires at least two hyperplanes positioned in the input space (by two hidden units at the hidden layer) and a hyperplane positioned in the hidden space (by the output unit detecting the *Mary* component).

Essentially, this reasoning uses the same analysis that was used to explain why perceptrons cannot solve the *exclusive-or (XOR)* task (Minsky & Papert, 1990; Hertz et al., 1991).

The positioning of hyperplanes in the input space is depicted in Figure 4.20. The necessity of two hyperplanes is a consequence of a first-order architecture and is independent of the representations of the question and pair vectors. For, although the values of these vectors are not known, the relationship between these vectors is known. The difference in the internal representations at the linear hidden layer (see Figure 4.19) between the (Q_1,(*Mary*,X)) case and the (Q_2,(*Mary*,X)) case is just a constant vector $\vec{W}_I\,(\vec{Q}_1 - \vec{Q}_2) = \vec{C}$. Similarly, the difference between the (Q_1,(X,*Mary*)) case and the (Q_2,(X,*Mary*)) case is also the vector $\vec{C}$.

Assuming that these two hyperplanes are in correct position the resulting representation at the hidden layer for these four types of points, dependent on whether they are on the high or low side (relative to some threshold) of these two hyperplanes, is shown in Figure 4.21. In the hidden unit activation space, these four point types must be partitioned by a single hyperplane into two groups: *Mary* (circles) and *not-Mary* (squares). Clearly, such a partitioning is possible. However, in addition, the network is required to demonstrate strong systematicity on the

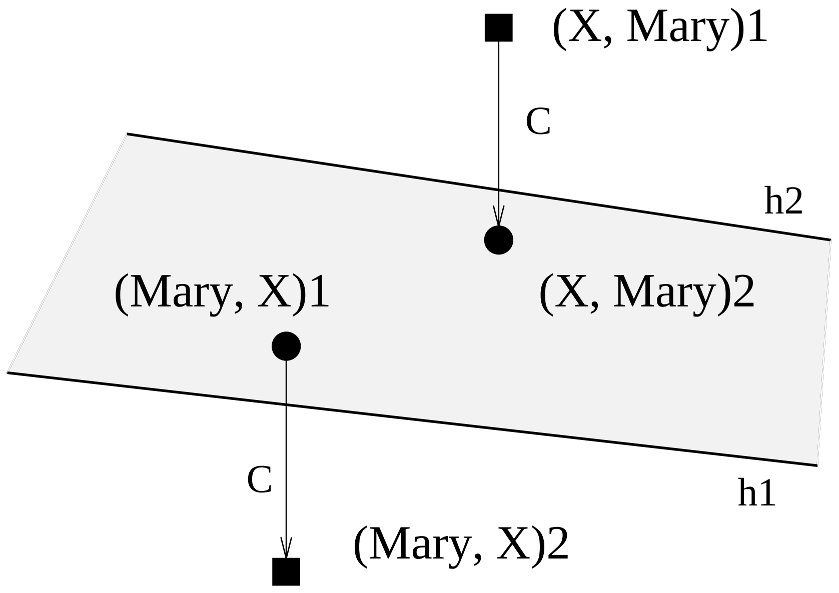


Figure 4.20: Orientation of the hidden unit hyperplanes to extract the *Mary* component. To implement the function there must be at least two hyperplanes in correct orientation in the input space to extract the *Mary* component. Circles indicate *Mary* is the target output, and squares indicate *Mary* is not the target output. Numerals indicate a request for the first or second component.

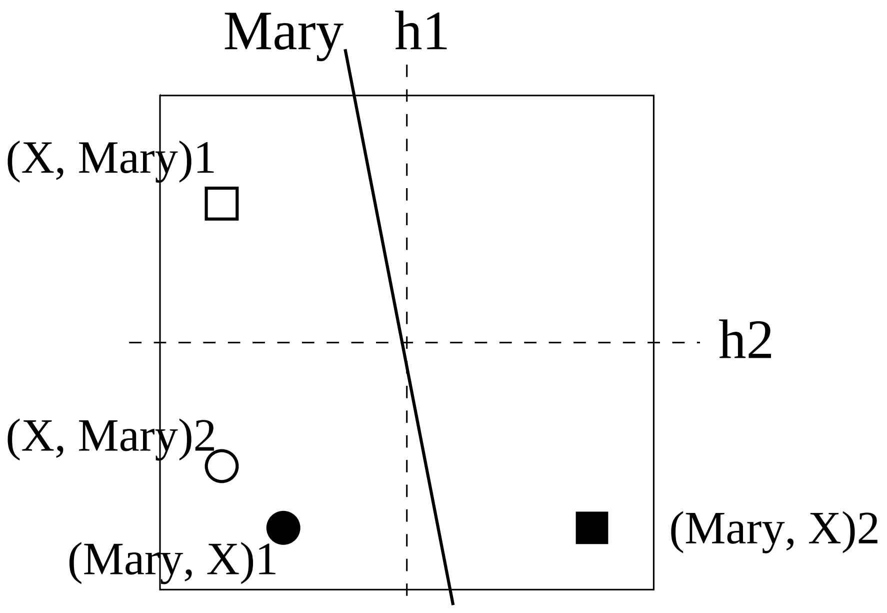


Figure 4.21: Orientation of the output unit hyperplane in the hidden unit activation space. Circles indicate *Mary* is the target output, and squares indicate *Mary* is not the target output. Solid circles and squares indicate training points and empty circles and squares indicate test points.

Mary component. Therefore, in the training set, one of the positions (e.g., *Mary* in the second position) does not occur (empty circle and square). Consequently, the training set does not provide any information regarding the partitioning of points along the vertical dimension, and therefore, the network cannot be expected to generalize to these cases.

Thus, as with the buffer solution to the auto-association task, there are two sets of dimensions: one encoding components in the first position, and one encoding components in the second position. The critical difference in the inference task is that at a single time step the network is required to recover either the first or second component depending on the question vector, whereas in the auto-association task at one time step only the first component was required with the second component recovered in the next time step. Consequently, the network solution to the task requires two output subspaces (in contrast with the solution to the representation task where only one output subspace is necessary) implemented by two sets of weights (Figure 4.22). Since the two sets of weights are independent the network,

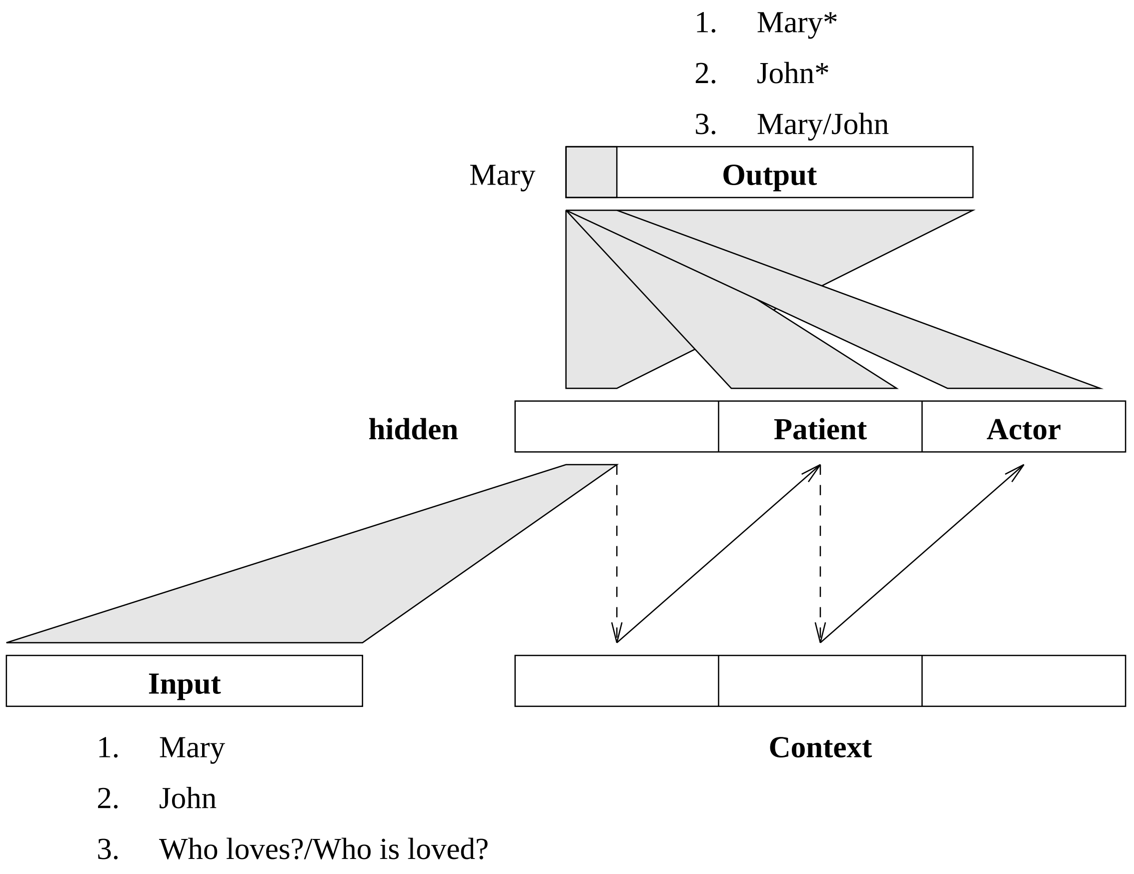


Figure 4.22: Weight and representation configuration resulting in weak systematicity of inference. Shaded regions and solid arrows indicate modified weights. Dashed arrows indicate fixed weights. Starred output indicates auto-association of input to assist in the formation of internal representations.

in general, must see components in both positions and therefore cannot demonstrate strong systematicity.

4.4.3 Other first-order three-layer networks

The weak systematicity result of the simple recurrent network does not make any assumptions about the nature of the internal representations of the ordered pairs. The assumptions made were that the external representations of components are encoded locally, and there is an intermediate layer of first-order units between the inputs and the outputs (which are also first-order units). Therefore, the result can also be applied to other architectures conforming to these restrictions.

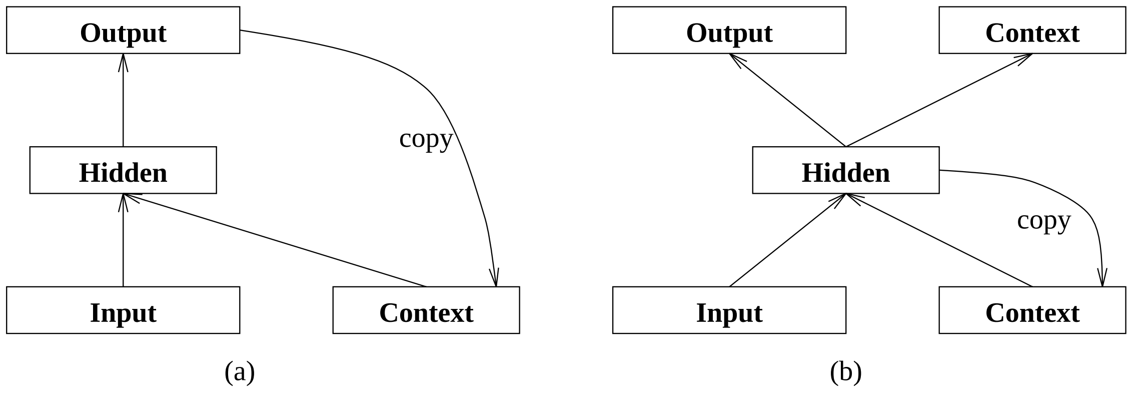


Figure 4.23: Two alternative recurrent network architectures: Jordan's recurrent network (a) and Pollack's recursive auto-associative memory (b).

Feedforward network

As mentioned previously, the context units in the simple recurrent network can be regarded as other input units holding a representation of the ordered pair. The two sets of input units map representations to hidden units which in turn map representations to output units. Since no assumptions are made regarding the nature of the order pair representation, three-layered feedforward networks are weakly systematic of inference with respect to this task.

Jordan's recurrent network

Jordan's (1990) recurrent network (see Figure 4.23(a)) differs from the simple recurrent network in that context is copied from the output unit activations rather than the hidden unit activations of the previous time step. Since the result makes no assumptions regarding the nature of the context representations the weak systematicity of inference result applies to the Jordan recurrent network.

Pollack's recursive auto-associative memory

Pollack's (1990) recursive auto-associative memory (see Figure 4.23(b)) is essentially the same as the simple recurrent network except there is an additional set of output units whose targets are the current context values (i.e., the network not only auto-associates the current input, but also the current context). However, these units (and their associated weights) play no part in the extraction of components. Their purpose is to encourage the formation of useful representations at the context layer. However, as mentioned above, the result of the simple recurrent network is independent of the actual representations at the context layer. Therefore, this architecture cannot demonstrate strong systematicity of inference on this task.

Forced simple recurrent network

The forced simple recurrent network (Maskara & Noetzel, 1992) is a variation of the recursive auto-associative memory in that the targets for the output units are the previous context, the current input and the next input. Additional information (hints) regarding the target function can improve learnability in networks (Abu-Mostafa, 1990; Suddarth & Kergos, 1990; Al-Mashouq & Reed, 1991; Suddarth & Holden, 1991). For example, use of hints allowed the simple recurrent network to discriminate input values appearing further back in time (Phillips & Wiles, 1991). In fact, auto-association of the inputs at the first and second time steps in the auto-association of 2-tuples task is an example of hint information except that the hints were on the same output units but were not considered for network evaluation. However, as with the recursive auto-associative memory, the additional output units play no part in the extraction of components in the forced simple recurrent network. Therefore, the network cannot demonstrate strong systematicity of inference on this task.

4.4.4 Architectural issues

The problem with the simple recurrent network and the other first-order three-layer networks is that the components must be extracted from one of two independent subspaces, depending on whether the query vector requests the first or second component. As in the case of the feedforward network, because there is an independence between the weights that implement these two mappings the network must be trained on all components in both positions.

The simple recurrent network was able to demonstrate strong systematicity of representation on the auto-association task because the task only requires the network to extract the first component at the third time step, with the second component being extracted at the fourth time step. At either time step the same subspace can be used to map internal component representations to outputs. Therefore, the same set of weights were can be used to implement the mapping. In this case, dependency was incorporated into the architecture as the component mappings were implemented by the same set of weights.

Dependency between component-access weights

The problem that strong systematicity of inference poses for an architecture is that all components must be simultaneously represented on different subspaces, which means there must be a different set of weights to implement the mappings from those subspaces to the output units. Therefore, to demonstrate strong systematicity there must be a dependency between these sets of weights. The representation task permitted all components to be accessed via a common subspace in the case of the simple recurrent network. Thus, dependency was incorporated as the network could use the same set of weights to access all components. However, the inference task does not permit all components being accessed from a common subspace in the case of these networks.

One way to incorporate a weight dependency is to collapse the various component subspaces onto a single common subspace from which all components, independent of their position within the complex object, can be extracted to the output. Thus, strong systematicity of inference is achieved as only one set of weights implements a position-independent mapping from a common internal subspace to the output, just as was suggested with the simple recurrent network in the auto-association task. The problem then is to incorporate a mechanism into a Connectionist architecture that, when given a query vector, returns the corresponding component subspace to the common subspace (Figure 4.24).

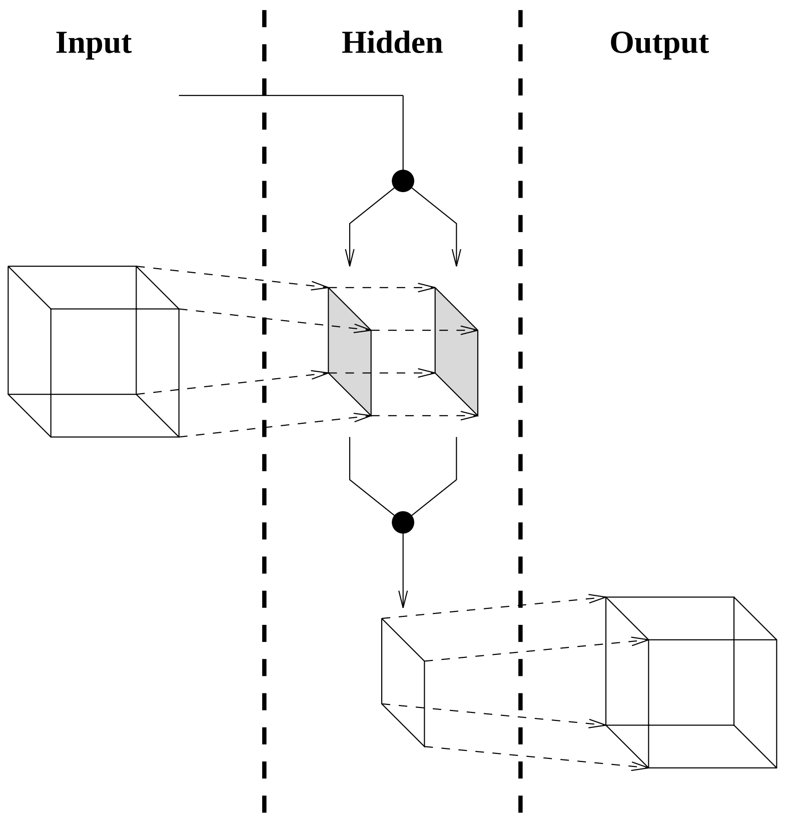


Figure 4.24: Characterization of the representational organization sufficient for strong systematicity of inference.

Tensors

One possibility is with the use of a tensor, rather than the more common vector representation space. Briefly, an $M \times N$-dimensional tensor space (constructed from two vector spaces of dimensionality M and N, respectively) can be conceptualized as M N-dimensional subspaces (or similarly, N M-dimensional subspaces). The inner product operator, provided in tensor calculus, effectively collapses the multiple subspaces down to a single subspace. The particular subspace selected depends on the vector with which the inner product was performed. Thus, tensors provide multiple representational subspaces, which can be indexed (or accessed) by other vectors.

The use of a tensor representational space in a Connectionist learning architecture is investigated in the next chapter with the purpose of addressing strong systematicity of inference in the querying of 2-tuples task.

4.5 Summary and conclusion

In this chapter, Hadley's strong systematicity was used as the criterion by which a model is regarded as exhibiting systematicity. The purpose of this chapter was: (1) to determine which models, if any, meet this criterion; and (2) to elucidate the properties which give rise to the model's degree of systematicity. Two tasks were designed to test whether Connectionist models could meet Hadley's strong systematicity criterion. They were: (1) auto-association of 2-tuples, designed to test strong systematicity of representation; and (2) querying of 2-tuples, designed to test strong systematicity of inference. A summary of the conclusions drawn from an evaluation of networks and their degrees of systematicity with respect to these two tasks is given in Table 4.2.

The first model examined was the feedforward network. It was shown not to be capable of exhibiting strong systematicity of representation. Essentially, the lack of strong systematicity is because of the independent relationship between the weights which implement the various component mappings. The simple recurrent

Table 4.2: Summary of the degrees of systematicity of networks with respect to the systematicity of representation task (auto-association of 2-tuples), and the systematicity of inference task (querying of 2-tuples). The networks examined included the feedforward network (FFN); simple recurrent network (SRN); and several first-order recurrent networks (RN*), including the simple recurrent network; Jordan's recurrent network; Pollack's recursive auto-associative memory; and Maskara's forced simple recurrent network.

Network	Task	Degree	Property
FFN	Represent.	Weak	Independent weights
SRN	Represent.	Strong	Dependent weights
FFN	Inference	Weak	Independent access weights
RN*	Inference	Weak	Independent access weights

network incorporates a weight dependence as all components are presented (and extracted) at the same set of input (and output) units. Therefore, the function which performs this mapping is implemented, in part, over the same set of weights. Simulation results confirmed that the simple recurrent network could exhibit strong systematicity of representation. Analysis of internal representations for the simple recurrent network suggested that the network learnt to organize its internal representations so that components were mapped from the input to a common internal subspace, and mapped from another common internal subspace to the output. Thus, dependency between component mapping weights was achieved as the network learnt to use the same set of weights.

The simple recurrent network was then examined on the second task designed to test strong systematicity of inference. Simulations showed that although the network generalized to novel combinations it did not demonstrate any degree of generalization across position on this task. The lack of strong systematicity suggested that the network did not have the right architectural bias to exhibit strong systematicity. By analysis of internal representations it was shown that the simple recurrent network cannot demonstrate strong systematicity on this task. The inference task demands that the components be extracted from two component subspaces. The independence of the weights that implement these two mappings meant the network must be trained on each component in both positions. It was also shown that a number of other first-order three-layer networks (i.e., one hidden layer) could not exhibit strong systematicity of inference on this task. Those networks included: the three-layer feedforward network; Jordan's recurrent network; Pollack's recursive auto-associative memory; and Maskara's forced simple recurrent network. The architectural bias lacking in these models was a dependency between component access weights.

Finally, it was suggested that dependency between component access weights could be achieved by collapsing the various component subspaces onto a single common subspace from which a position independent mapping could be implemented by a single set of weights. Thus, weight dependency is incorporated by using the same set of weights. It was suggested that this bias could be achieved in a tensor representation scheme using the inner product operator. In the next chapter, details of a Connectionist network incorporating these ideas is presented and tested by simulation on a querying of 3-tuples task.

Chapter 5

Strong systematicity of inference: The Tensor-Recurrent Network

5.1 Introduction

In the previous chapter, a number of Connectionist models were analyzed for their capacity to exhibit strong systematicity. The importance of Hadley's work was to identify and specify a degree of generalization that is clearly evident in people, but has yet to be demonstrated in Connectionist models. The main results of the previous chapter were to show that the three-layer feedforward network and several first-order three-layer recurrent networks cannot demonstrate strong systematicity of inference with respect to the querying of 2-tuples task.

The characteristic property that prevented these models from exhibiting strong systematicity of inference was an independence between the weights that access internal component representations from the two component subspaces. It was then suggested that a mechanism that incorporated such a dependence would be sufficient to demonstrate strong systematicity of inference. The use of tensor representations was suggested as one possible mechanism. In this chapter, tensor representations are used as the basis for a Connectionist architecture, called the tensor-recurrent network, for the purpose of exhibiting strong systematicity of inference with respect to a querying of 3-tuples task.

Before presenting the tensor-recurrent network, the concept of tensor representations, and their Connectionist implementation (introduced by Smolensky, 1987b) is reviewed in section 2. The use of tensors for representing complex objects assumes the existence of a number of representational components. This assumption raises an issue, which is discussed in section 3, regarding the origins of these representations. In section 4, it is shown how these representational components are learnable in the tensor-recurrent network by combining a tensor representation architecture with a recurrent network learning architecture. The tensor-recurrent network is then evaluated for strong systematicity of inference on a querying of 3-tuples task in section 5. Finally, a summary and conclusion are given in section 6.

5.2 Tensor representations

Tensor representations were introduced by (Smolensky, 1987b) as a means of representing complex objects in a Connectionist architecture. As already shown in chapter 2, complex objects can be represented as the sum of the outer products of **component-role** vector representation pairs. The outer product ($\vec{T}$) of two vectors ($\vec{V}$ and $\vec{W}$) is defined as:

$$\vec{T} = \vec{V} \otimes \vec{W} = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} \begin{pmatrix} w_1 & \cdots & w_n \end{pmatrix} = \begin{pmatrix} v_1 w_1 & \cdots & v_1 w_n \\ \vdots & & \vdots \\ v_m w_1 & \cdots & v_m w_n \end{pmatrix} \tag{5.1}$$

A Connectionist network that implements the outer product is given in Figure 5.1. In this network, connections have a fixed weight of value one. The activation of *tensor* unit T_{ij} at time step t is defined by the equation:

$$T_{ij}^t = v_i w_j + T_{ij}^{t-1}, \tag{5.2}$$

where v_i is the activation of the ith unit in the *vector(V)* group of units; w_j is the activation of the jth unit in the *vector(W)* group of units; and T_{ij}^{t-1} is the activation of the ijth *tensor* unit at the previous time step $(t-1)$. The multiplicative activation term makes the tensor an example of a higher-order network.

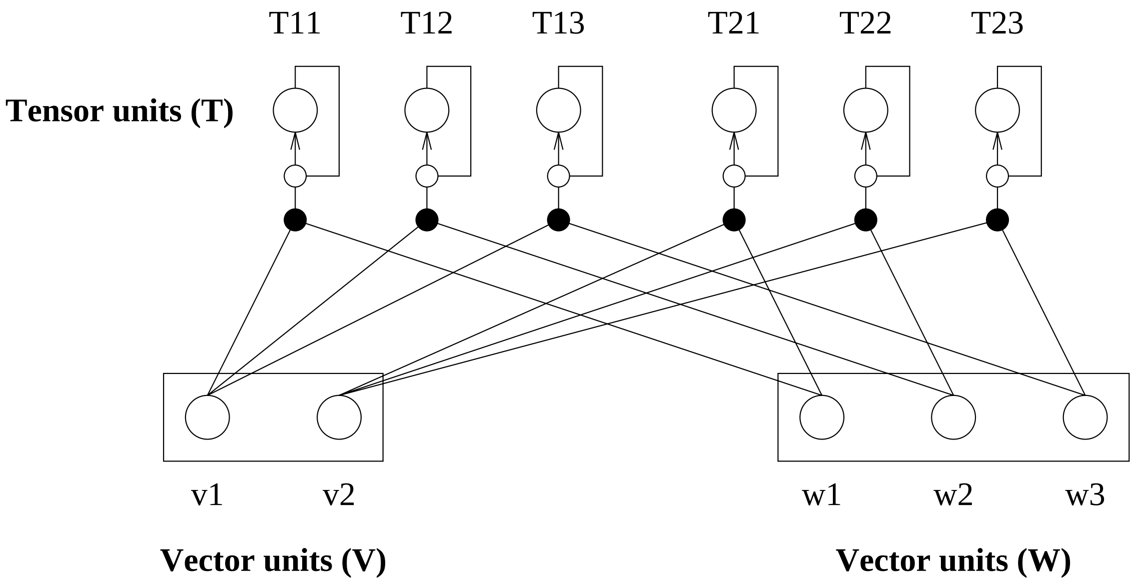


Figure 5.1: A Connectionist implementation of the outer product (T) of two vector representations (V and W). All connections are of fixed weight 1. Solid circles indicate the product of input activity, and empty circles indicate the sum of input activity. The activation of a tensor unit is the product of the activity of previous units plus the tensor unit's activation from the previous time step.

In addition to constructing complex representations via the outer product operator, component representations can be accessed via the inner product operator. The inner product of a tensor ($\vec{T}$) and a **cue** vector ($\vec{W}$) is defined as:

$$\vec{T} \odot \vec{W} = \vec{V} = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix}, \tag{5.3}$$

where $v_i = \sum_{j=1}^{n} T_{ij} w_j$; and n is the dimensionality of vector $\vec{W}$.

The Connectionist network that implements the inner product is given in Figure 5.2. In this network, connections have a fixed weight of value one. The activation of *product* unit p_{ij} is defined as:

$$p_{ij} = T_{ij} w_j, \tag{5.4}$$

where T_{ij} is the activation of the ijth *tensor* unit; and o_j is the activation of the jth unit in the *vector(W)* group of units. The activation of *vector(Vout)* unit v_{out_i}

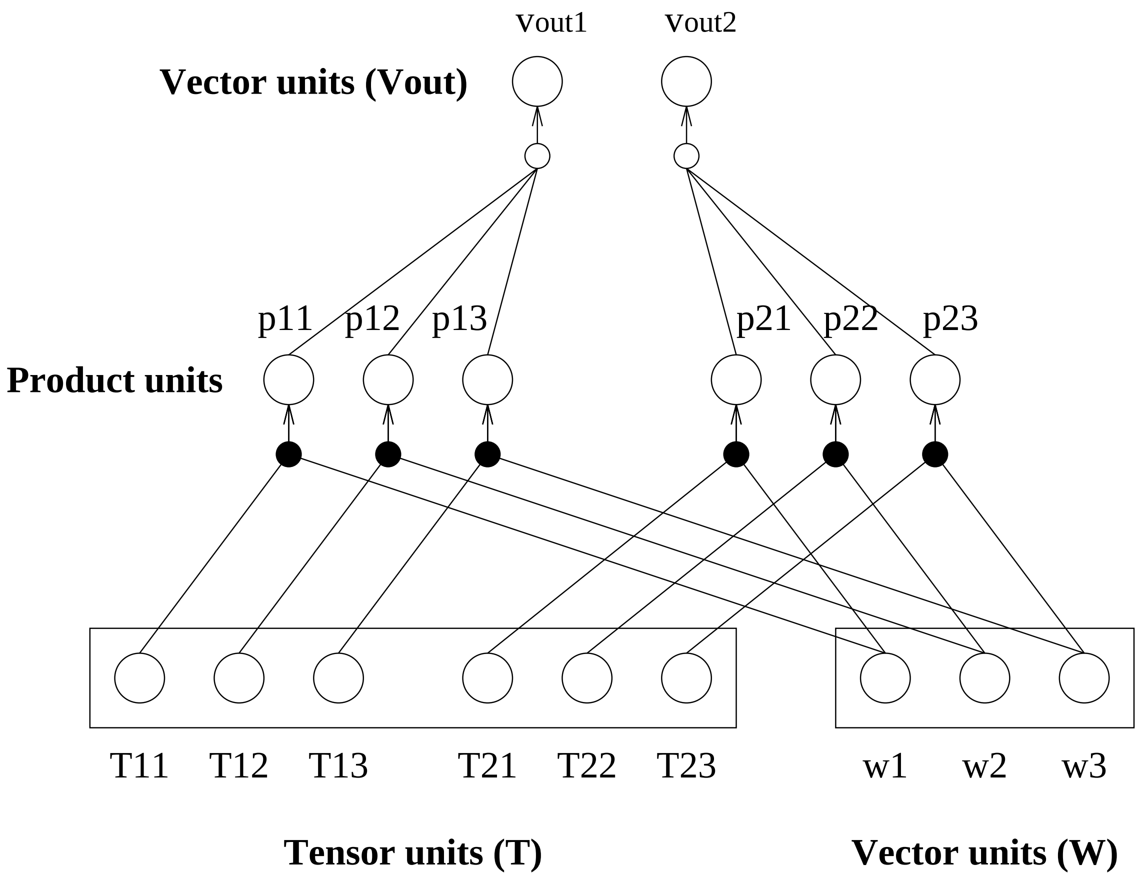


Figure 5.2: A Connectionist implementation of the inner product (Vout) of a tensor representation (T) and a vector representation (W). All connections are of fixed weight 1. Solid circles indicate the product of input activity, and empty circles indicate the sum of input activity. The activation of a product unit is the product of the activity of previous units. The activation of a vector(Vout) unit is the sum of the activity of previous units.

is defined as:

$$v_{out_i} = \sum_{j=1}^{n} p_{ij}, \tag{5.5}$$

where p_{ij} is the activation of *product* unit ij. The activation is summed over all n incoming connections[1].

The following is an example of how these two networks, together, can process examples from the querying of 2-tuples task, which was given in the previous chapter. Suppose the network is presented with the 2-tuple (John, Mary) and a query

[1]The number of incoming connections n is the dimensionality of the cue vector, which in this network implementation is the number of *vector(W)* units.

requesting the second component. Then the sequence of inputs and targets is:

Time	Input	Output
1.	John	–
2.	Mary	–
3.	Second	Mary.

The activation of the network units throughout this sequence is described by the following sequence of equations:

$$\begin{array}{rrcl}
0. & \vec{T}^0 & = & \vec{0} \\
1. & \vec{T}^1 & = & \vec{T}^0 + \vec{V}_{John} \otimes \vec{W}_{first} \\
2. & \vec{T}^2 & = & \vec{T}^1 + \vec{V}_{Mary} \otimes \vec{W}_{second} \\
3. & \vec{V}_{out} & = & \vec{T}^2 \odot \vec{Q}_{second} \\
 & & = & \vec{V}_{John} \otimes (\vec{W}_{first} \odot \vec{Q}_{second}) + \vec{V}_{Mary} \otimes (\vec{W}_{second} \odot \vec{Q}_{second}) \\
 & & = & \vec{V}_{Mary} \qquad (\text{when } \vec{W}_{first} \perp \vec{Q}_{second}, \text{ and } \vec{W}_{second} \odot \vec{Q}_{second} = 1).
\end{array}$$

The *tensor* units are initialized with zero activation. At time step 1, the two groups of units *vector(V)* and *vector(W)* (see Figure 5.1) are presented with representations of the first component ($\vec{V}_{John}$) and the first component's role ($\vec{W}_{first}$), respectively. The activation of the *tensor* units becomes their activation at the previous time step plus the outer product of the component and role vectors. At time step 2, *vector(V)* and *vector(W)* units are presented with representations of the second component ($\vec{V}_{Mary}$) and the second component's role ($\vec{W}_{second}$), respectively. The *tensor* units are updated as in the previous time step. At time step 3, a representation of the query ($\vec{Q}_{second}$), or cue vector, is presented to the *vector(W)* units. The activation of the *vector(Vout)* units are a result of the inner product of the activations of the *tensor* units and the *vector(W)* units. If the representation of the first component ($\vec{V}_{John}$) is orthogonal to the representation of the query ($\vec{Q}_{second}$), and if the inner product of the second component ($\vec{V}_{Mary}$) and the query vector is one[2], then the representation of the second component will be correctly extracted.

[2]This situation occurs, for example, when the two vectors are equal and of length one.

5.3 Representational issues

The use of tensors to represent and process complex objects assumes the existence of external agents for generating appropriate component (e.g., $\vec{V}_{John}$), role (e.g., $\vec{W}_{first}$) and cue (e.g., $\vec{Q}_{first}$) representations. In this section, the assumptions regarding each of these three aspects of tensor representations are discussed.

5.3.1 Component representations

In the tensor scheme, component representations are bound (by an outer product) to a representation of their associated roles. If the appropriate cue vector is applied (by an inner product) to the tensor representation then the original component representation is extracted. Thus, in a tensor scheme the input and output representations of a component object are the same. However, the input representation of the object *John*, for example, could be a string of letters (in the case where one is receiving visual input), whereas the output representation of the same object could be a sequence of phonemes (in the case where one is sending auditory output). Therefore, it must be assumed that there exists additional agents that perform the mappings from an external input representation of an object to an internal representation (which is used in the tensor construction), and from the internal representation to an external output representation of the same object (which may differ from the external input representation).

5.3.2 Role representations

In a tensor representation of complex objects, each component representation is bound to a representation of the component's role within the complex object. In using tensor representations it is assumed that there exists some other agent that generates role representations for each component, since these representations are not explicitly represented in the input[3]. That is to say, there is not an input rep-

[3]Although Smolensky (1987a) showed how "optimal" role representations could be learnt with his *recirculation* algorithm, these roles are learnt in isolation (i.e., separate from any binding to

resentation denoting *patient*, for example, in the sentence *John loves Mary.* Roles must be deduced from other information such as relative positioning of components. In addition, role representations must be orthogonal to each other for the component representations to be subsequently extracted, via the inner product operator, without interference. For example, suppose vectors $\vec{V}$ and $\vec{W}$ are bound to roles $\vec{R}_1$ and $\vec{R}_2$, respectively. Then, the extraction of the first component V using $\vec{R}_1$ as the cue vector results in:

$$(\vec{V} \otimes \vec{R}_1 + \vec{W} \otimes \vec{R}_2) \odot \vec{R}_1 = \vec{V} + \alpha\vec{W}, \qquad \text{where } \vec{R}_1 \odot \vec{R}_2 = \alpha$$

In the case where the roles vectors are of unit length, α is the component of $\vec{R}_1$ in the direction of $\vec{R}_2$. The choice of role vectors is important for the successful extraction of component representations. Thus, a tensor scheme assumes the existence of agents that generate appropriate role vectors.

5.3.3 Cue representations

Having constructed tensor representations of complex objects, component objects can be extracted via the inner product operator. Typically, the cue vector is chosen, by some external agent, to be the same as the role vector to which the desired component was composed. However, since the role vectors were generated internally there is no reason to expect that an external representation of a query will be the same as the internal representation of the role. For example, in the complex object *John chased Mary*, the component *John* is bound to a representation of the *agent* role. It is assumed that the query *Who chased?* gets mapped to the same representation of agent, at which point the *John* component can be extracted from the tensor representation using the inner product operator.

component vectors). No demonstration was given as to how the recirculation algorithm could be used in any task. The recirculation algorithm could have been used to train a three-layer feedforward network to auto-associate N-tuples if one assumes that the activation function for the hidden units is the identity function, and that the input to hidden weight matrix equals the transpose of the hidden to output weight matrix (i.e., $W_{I,H} = W_{H,O}^T$). However, even with these assumptions there is still an independence between the weights that access components from different positions, and therefore the network will not demonstrate strong systematicity with respect to this task.

Tensor representational schemes were introduced by Smolensky as a way of implementing the capacity to represent complex objects in a Connectionist architecture. In chapter 2, it was argued that a "potential" contribution of Connectionism is in an explanation of the acquisition of systematic behaviour. If tensor representations are to prove useful in providing such an explanation then this issue regarding the origins of component, role and cue representations must be addressed. In the next section, it is shown how these representations can be learnt with the tensor-recurrent network.

5.4 The tensor-recurrent network

In the previous chapter, a number of Connectionist networks were analyzed for their capacity to exhibit strong systematicity. Characteristic of these networks was the use of a vector space to represent complex objects. Although these networks can represent most functions, it was shown that there was not sufficient architectural bias to account for strong systematicity. Since tensors have been shown capable of representing complex objects, a network incorporating tensor representations was suggested as a way of addressing strong systematicity. However, to this point, there has not been an effective procedure for learning tensor representations. Consequently, assumptions must be made regarding the nature and origins of component, role and cue representations. The motivation behind the tensor-recurrent network was to incorporate a learning mechanism with a tensor representation scheme so that component, role and cue representations are learnt as a consequence of the demands of the task, rather than provided by external agents. The following is a description of the tensor-recurrent network and how it addresses the issue of acquiring component, role and cue representations.

The tensor-recurrent network is depicted in Figure 5.3. The network has many of the components that are characteristic of the recurrent networks of the previous chapter. That is, there are input and output units for presenting and extracting component representations, and there are units for storing activations from the previous time step. The main difference with the tensor-recurrent network is in the

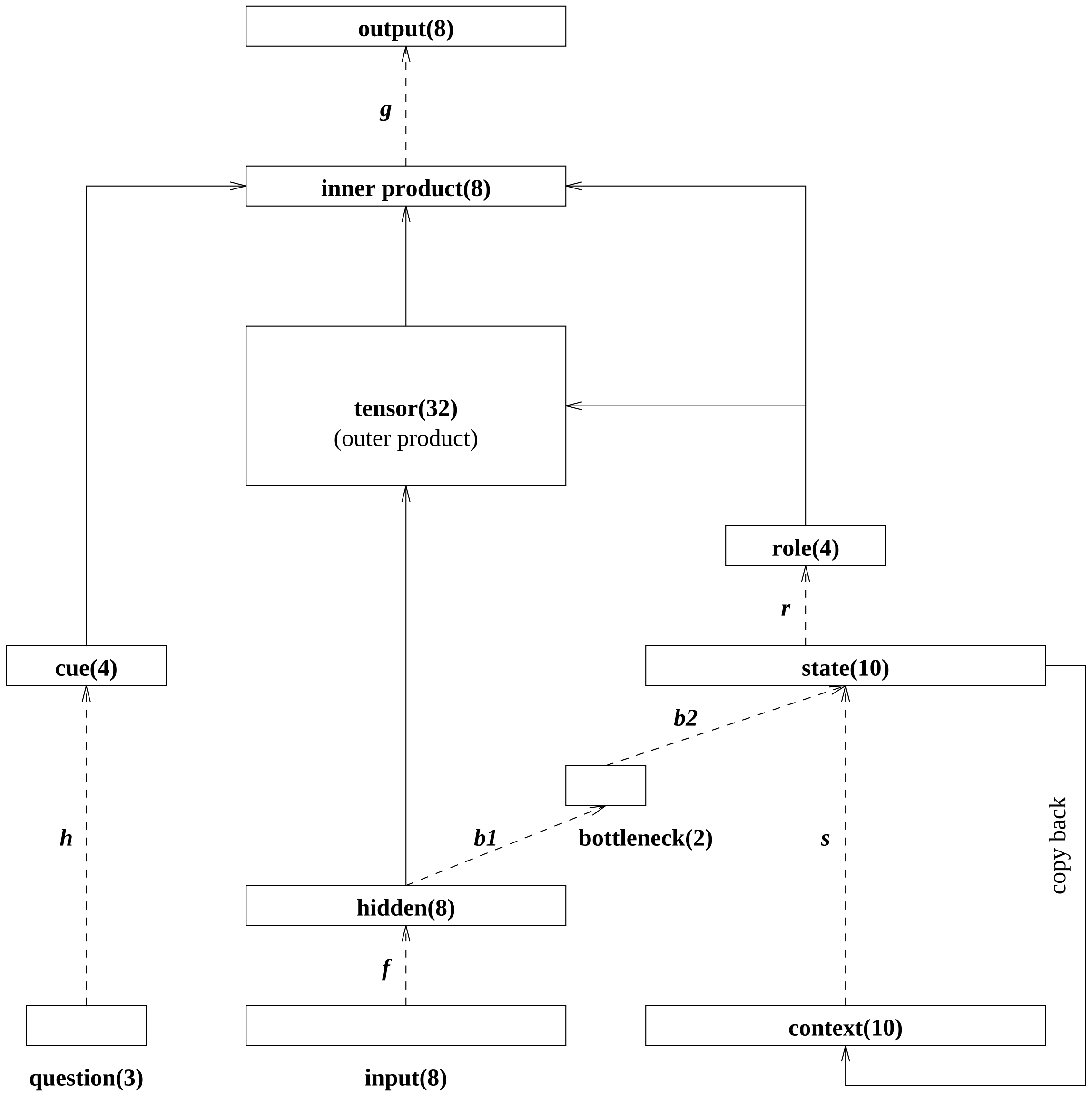


Figure 5.3: The tensor-recurrent network architecture. Dashed arrows indicate completely connected modifiable weights and their destination units (which are the *cue*, *hidden*, *bottleneck*, *state*, *role*, and *output* units) have *tanh* as their activation function. Solid arrows indicate fixed connections of weight 1, and the connectivity between their source and destination units is such as to implement the operator as shown (i.e., inner product, outer product, and copy). Parenthesized values indicate number of units used in simulations.

separation of the responsibility for representing component and role information into two sets of units: (1) *hidden* units, which construct internal representations of component objects; and (2) *role* units, which construct internal representations a component's role within a complex object. This organization contrasts with the recurrent networks where a single set of *hidden* units is responsible for the representation of both component and role information. Before discussing the implications of this difference with respect to demonstrating strong systematicity, the internal behaviour of the network and its capacity to learn component, role and cue representations is presented in detail.

5.4.1 Architecture

The description of the tensor-recurrent network has two parts: (1) an outline of the functionality of each group of units; and (2) the activation functions and connectivity of weights that implement the functionality of each group of units.

Unit functionality

The functionality of each group of units is organized into three major categories: (1) external representation units; (2) tensor-related units; and (3) feedback-related units. Their functionality is as follows:

1. **External representation units** The purpose of these units is to hold the input and output unit activation. The three types of external units are:
   - *Input*: external input representation of component objects.
   - *Question*: external representation of a question.
   - *Output*: external output representation of component objects.
2. **Tensor-related units**. These units are concerned with implementing the inner and outer product operators. The five types of units associated with these two operators are:
   - *Tensor*: tensor representations of complex objects.

- *Hidden*: internal representations of component objects.
- *Role*: internal representations of a component's role within a complex object.
- *Cue*: internal representations of vectors for accessing component representations within the tensor representations.
- *Inner product*: extracted component representations.

3. **Role-related units** These units are concerned with creating new role representations at each time step. The three types of units are:
   - *State*: hold the current state vector.
   - *Context*: holds the activation of the *state* units from the previous time step.
   - *Bottle-neck*: an intermediate representation between the *hidden* units and the *state* units.

Implementation

The activation of each *context* unit is defined as:

$$c_i^t = s_i^{t-1}, \tag{5.6}$$

where c_i^t is the activation of *context* unit i at time step t; and s_i^{t-1} is the activation of *state* unit i at the previous time step $t-1$.

The activation of each *tensor* unit is defined as:

$$T_{ij}^t = h_i \, r_j + T_{ij}^{t-1}, \tag{5.7}$$

where T_{ij}^t and T_{ij}^{t-1} are the activations of *tensor* unit ij at time steps t and $t-1$, respectively; h_i is the activation of *hidden* unit i; and r_j is the activation of *role* unit j.

The activation of each *inner product* unit is defined as:

$$ip_i = \sum_{j=1}^{n} T_{ij}(q_j + r_j), \tag{5.8}$$

where ip_i is the activation of *inner product* unit i; q_j is the activation of *cue* unit j; r_j is the activation of *role* unit j; and n is the number of *cue/role* units.

The activation of all other units, which includes the *cue*, *hidden*, *bottleneck*, *state*, *role*, and *output* units, is defined as:

$$o_j = \tanh(\sum_{i=1}^{k} x_i w_{ij} + b_j), \tag{5.9}$$

where o_j is the activation of unit j; x_i is the activation of a preceding unit i; w_{ij} is the connection weight from unit i to unit j; b_j is the bias term on unit j; and k is the number of incoming connections.

Each dashed arrow indicates complete connectivity between source and destination layers (i.e., every unit in the source layer is connected to every unit in the destination layer). The weights associated with these connections are modifiable through a learning algorithm. The solid lines indicate unmodifiable weighted connections. The connectivity between layers is such as to implement the following functions:

- **Outer product** The connectivity from the *hidden* and the *role* units to the *tensor* units implements the outer-product $\vec{h} \otimes \vec{r}$, where $\vec{h}$ is a vector at the *hidden* units providing an internal representation of some component, and $\vec{r}$ is a vector at the *role* units providing an internal representation of the component's role. An example of the connectivity that implements the outer product was given in Figure 5.1.

- **Inner product** The connectivity from the *tensor* units and the *role* units to the *inner product* units implements the inner product $\vec{T} \odot \vec{r}$, where $\vec{T}$ is a tensor representation at the *tensor* units of some complex object and $\vec{r}$ is a vector representation at the *role* units. An example of the connectivity that implements the inner product was given in Figure 5.2. The connectivity between *cue* and *inner product* units implements the same function, except that the inner product is: $\vec{T} \odot \vec{q}$, where $\vec{q}$ is a vector representation at the *cue* units.

Table 5.1: Order of activation for the tensor-recurrent network when given the sequence *John*, *Mary*, *first*.

Units	$T = 1$	$T = 2$	$T = 3$
Input	$\vec{V}_{John}$	$\vec{V}_{Mary}$	$\vec{0}$
Question	$\vec{0}$	$\vec{0}$	$\vec{Q}_{first}$
Hidden	$f(\vec{V}_{John})$	$f(\vec{V}_{Mary})$	$\vec{H}_{bias}$
Cue	$\vec{C}_{bias}$	$\vec{C}_{bias}$	$h(\vec{Q}_{first})$
Context	$\vec{S}_0$	$\vec{S}_1$	$\vec{S}_2$
Bottle	$\vec{B}_1$	$\vec{B}_2$	$\vec{B}_3$
State	$\vec{S}_1$	$\vec{S}_2$	$\vec{S}_3$
Role	$\vec{R}_1$	$\vec{R}_2$	$\vec{R}_3$
Tensor	$\vec{T}_1 = \vec{T}_0 + f(\vec{V}_{John}) \otimes \vec{R}_1$	$\vec{T}_2 = \vec{T}_1 + f(\vec{V}_{Mary}) \otimes \vec{R}_2$	$\vec{T}_3 = \vec{T}_2 + \vec{T}_{noise}$
Inner	$\vec{T}_1 \odot \vec{R}'_1$	$\vec{T}_2 \odot \vec{R}'_2$	$\vec{T}_3 \odot h(\vec{Q}_{first})'$
Output	$g(\vec{T}_1 \odot \vec{R}'_1)$	$g(\vec{T}_2 \odot \vec{R}'_2)$	$g(\vec{T}_3 \odot h(\vec{Q}_{first})')$

- **Copy** The connectivity between *state* and *context* units implements a copy-back function, which is the same as in Elman's (1990) simple recurrent network (i.e., one-to-one connections, with a fixed weight of one).

Those units that have modifiable weights also have a modifiable bias, which can be implemented as an incoming weighted connection whose activity is always one. Other units do not have a bias.

An example of activity propagation

An example sequence from the querying of 2-tuples task is used to show the processing of activation in the tensor-recurrent network. The sequence *John*, *Mary*, *first* will result in the following sequence of activation, which is summarized in Table 5.1.

At the first time step ($T = 1$), the tensor-recurrent network is presented with an external input representation ($\vec{V}_{John}$) of the component *John*. At this point, the network has not been queried so the activation at the *question* units is zero. This representation is mapped (via function f) to the *hidden* units to form the internal representation $f(\vec{V}_{John})$. The *hidden* unit representation is mapped (via function b_1) to the *bottleneck* units, resulting in vector $\vec{B}_1$, and from the *bottleneck* units to the *state* units via function b_2. At the same time, state vector $\vec{S}_0$ at the *context* units is mapped via function s to the *state* units. The *state* units take the results of these two functions and produce a new state vector $\vec{S}_1$. The new state vector is mapped (via function r) to the role vector $\vec{R}_1$, which is a representation of the role of the first component object. The outer product of vectors $f(\vec{V}_{John})$ and $\vec{R}_1$ is added to the previous tensor ($\vec{T}_0$) to give a new tensor representation $\vec{T}_1$ at the *tensor* units. The inner product of the tensor and the role plus cue vectors is then performed. Although the *question* units have zero activity at the first time step, and therefore, do not pass on any activity to the *cue* units, the bias terms associated with the *cue* units result in an activity vector $\vec{C}_{bias}$. Consequently, the activity vector at the *inner product* units is $\vec{T}_1 \odot \vec{R}'_1$, where $\vec{R}'_1 = \vec{R}_1 + \vec{C}_{bias}$. The activity vector at the *inner product* units is subsequently mapped (via function g) to the *output* units, resulting in the output vector $g(\vec{T}_1 \odot \vec{R}'_1)$. In addition, the state vector is propagated back to the *context* units in preparation for the next time step.

Activity vectors are propagated, in a similar manner, during the second time step. At the third time step, the network is presented with the question requesting the first component, which is represented as the vector $\vec{Q}_{first}$. The question vector is mapped (via function h) to form an internal query vector $h(\vec{Q}_{first})$ at the *cue* units. Although the activation at the *input* units is zero at the third time step, the bias terms at the *hidden* units (i.e., vector $\vec{H}_{bias}$) and the state vector $\vec{S}_2$ are propagated forward to generate a new state vector $\vec{S}_3$, and subsequently a new role vector $\vec{R}_3$. The outer product of the *hidden* and *role* unit vectors results in a noise term $\vec{T}_{noise}$ (i.e., a term that contains no information regarding the complex object (*John*,*Mary*)). Again, the inner product of the tensor vector and the role plus cue vectors is performed. The inner product results in the vector $\vec{T}_3 \odot h(\vec{Q}_{first})'$, where $h(\vec{Q}_{first})' = h(\vec{Q}_{first}) + \vec{R}_3$. Finally, this vector is mapped to the *output* units resulting in the output vector $g(\vec{T}_3 \odot h(\vec{Q}_{first})')$.

Having traced through the propagation of activity for an example sequence, the next step is to show how this activity can be used to train the tensor-recurrent network to construct appropriate internal component, role and cue representations.

5.4.2 Learning

The output vector produced by the network as a result of the sequence of input vectors can then be compared to a target vector. In the case of the previous example, the target output at third time step is $\vec{V}^T_{John}$. Just as with the networks of the previous chapter, by incorporating an error function within the network, performance can be improved by backpropagating an error signal, which is a function of the network and target output vectors. Weights are adjusted as a function of the error signal so as to reduce network error. Again, it is the principle of error backpropagation (Rumelhart et al., 1986) that drives the construction of internal representations. The same principle is used in the tensor-recurrent network. The interesting feature of the tensor-recurrent network is that the error signal is backpropagated along the fixed weights and units that implement the tensor operators to the modifiable weights that generate component, role and cue representations. In this way, these representations can be learnt in response to the demands of the task, rather than pre-specified by some external agent. The following is an explanation of how the appropriate component, role and cue representations can be learnt given the principle of error backpropagation and the internal organization of the tensor-recurrent network.

Component representations

It was mentioned in section 5.3 that a tensor representation scheme assumes the existence of external agents that perform the mappings between external and internal component representations. This assumption is avoided in the tensor-recurrent network by providing a layer of modifiable weights before and after the *tensor-related* units (i.e., the weights associated with functions f and g, respectively).

Suppose that, in the (*John,Mary,first*) example, the network is also required to output the current object for the first and second time steps (as was done for the networks of the previous chapter to encourage the formation of useful internal representations). Then, using the sum of squares error function, the error term at the first time step is:

$$E = \|\vec{V}^T_{John} - g((f(\vec{V}_{John}) \otimes \vec{R}_1) \odot (\vec{R}_1 + \vec{C}_{bias}))\|,$$

where $\vec{V}_{John}$ and $\vec{V}^T_{John}$ are the input and target output representations of *John*, respectively; $\vec{R}_1$ is the associated role vector, and $\vec{C}_{bias}$ is a bias vector. Thus, error E goes to zero ($E \rightarrow 0$) when, for example, the following three conditions occur:

1. $\vec{C}_{bias} \rightarrow 0$;
2. $\|\vec{R}_1\| \rightarrow 1$; and
3. $g(f(\vec{V}_{John})) \rightarrow \vec{V}^T_{John}$.

That is, when the modifiable weights that implement the various mapping functions are adjusted so that these three conditions occur.

In the case where $\vec{R}_1 \odot (\vec{R}_1 + \vec{C}_{bias}) = 1$, which occurs, for example, when $\vec{C}_{bias} = 0$ and $\|\vec{R}_1\| = 1$, the vector representation at the *hidden* units is the same as the vector representation at the *inner product* units. That is,

$$f(\vec{V}_{John}) = (f(\vec{V}_{John}) \otimes \vec{R}_1) \odot (\vec{R}_1 + \vec{C}_{bias})$$

Consequently, the network reduces to a three-layer feedforward network, where any input to output mapping is representable (given sufficient hidden units), and therefore, "potentially"[4] learnable.

Role representations

Tensor representations also assume the existence of an external agent that generates orthogonal role vectors for each component. Referring to the (*John,Mary,first*)

[4]As with the feedforward network, there is no guarantee that a multiple layer network with non-linear units will learn an arbitrary function.

example, at the second time step, the error term is:

$$E = \|\vec{V}^T_{Mary} - g((f(\vec{V}_{John}) \otimes \vec{R}_1 + f(\vec{V}_{Mary}) \otimes \vec{R}_2) \odot (\vec{R}_2 + \vec{C}_{bias}))\|,$$

where $\vec{V}^T_{Mary}$ is the target output for the object *Mary*; and $\vec{R}_2$ is its associated role representation. Thus, error goes to zero when, for example, the following four conditions occur:

1. $\vec{C}_{bias} \rightarrow 0$;
2. $\|\vec{R}_2\| \rightarrow 1$;
3. $\vec{R}_1 \perp \vec{R}_2$; and
4. $g(f(\vec{V}_{Mary})) \rightarrow \vec{V}^T_{Mary}$.

Thus, the network learns appropriate role vectors by adjusting weights which minimize an error function that goes to zero when the role vectors are orthogonal.

Cue representations

At the third time step, the error term is:

$$\begin{aligned} E &= \|\vec{V}^T_{John} - g((f(\vec{V}_{John}) \otimes \vec{R}_1 + f(\vec{V}_{Mary}) \otimes \vec{R}_2 + f(\vec{H}_{bias}) \otimes \vec{R}_3) \\ &\quad \odot(h(\vec{Q}_{first}) + \vec{R}_3))\|, \end{aligned}$$

where $\vec{H}_{bias}$ is the bias vector at the *hidden* units; and $\vec{Q}_{first}$ is the representation of the query requesting the first component. At this time step, error goes to zero when:

1. $\vec{R}_3 \rightarrow 0$;
2. $\vec{R}_1 \perp \vec{R}_2 \perp \vec{R}_3$;
3. $\|\vec{R}_1\| \rightarrow 1$;
4. $\|\vec{Q}_{first} - \vec{R}_1\| \rightarrow 0$; and

5. $g(f(\vec{V}_{John})) \rightarrow \vec{V}^T_{John}$.

The error function drives the network to learn an appropriate cue vector that when applied to the tensor representation by the inner product operator results in an output representation of the first component *John*.

5.5 Evaluating the network

The lack of strong systematicity in the networks of the previous chapter was attributed to the independence between the weights that access components from different representational subspaces. A tensor representation scheme was suggested as it has the property that multiple subspaces can be collapsed down to a single subspace under the inner product operator. Thus, a weight dependency is introduced across component subspaces as all components, independent of their role, are mapped from the common internal subspace to the output. Therefore, learning to extract a component representation in one position (role) generalizes to component representations in other positions (roles). However, extraction of the correct component requires the network to learn the appropriate component, role and cue representations. In the tensor-recurrent network, this requirement means learning the weights that will generate these vectors. The solutions to the error functions (above) showed that these weights exist, however, they do not indicate whether they will necessarily be learnt. For example, the error surface could contain local minima preventing the network from learning a set of weights that generates strong systematic behaviour. The purpose of this section is to test, through simulation, the tensor-recurrent network's capacity to learn a set of weights that results in strong systematicity on an inference task.

5.5.1 Querying of 3-tuples task

The task used in this section is based on simple sentences conforming to the structure *agent-action-patient*. Each component is presented to the network one per time step after which the network is given one of three possible questions: *Who* *performed the action?*; *What was the action?*; and *Who was effected by the action?*, from which the network should respond with the *agent*, *action*, and *patient* components, respectively. For example, the sentence-question pair *"John loves Mary. Who performed the action?"* would be represented by a sequence of four vectors (representing *John*, *loves*, *Mary*, *Who performed the action?*, respectively) presented in that order to the network. At the fourth time step the correct response is a vector representing *John* at the output layer. This task is similar to the querying of 2-tuples task of the previous chapter, except that the name of the binary relation (e.g., *loves*) is included in the input.

The *action* and *patient* components are drawn from the set {*Bill*, *John*, *Karen*, *Mark*, *Vivian*}, and the *action* component is drawn from the set {*calls*, *chases*, *loves*}. That is, there are 5 (agents) $\times$ 3 (actions) $\times$ 5 (patients) $\times$ (3 questions) = 225 possible sentence-question pairs. All components are encoded locally (i.e., by orthogonal vectors where one dimension has a value of 1 and the rest 0). The network is said to have demonstrated strong systematicity if having only seen *Mary* in the *agent* position in the training set, for example, the network consistently (above chance level) generalizes to cases where *Mary* is in the *patient* position in the test set.

In the next subsection, simulations are run on the tensor-recurrent network to test whether the network will exhibit strong systematicity of inference on this task.

5.5.2 Simulations

The number of units used in these simulations was given in Figure 5.3. The simulation conditions for the tensor-recurrent network were as follows:

- Local encoding of input and output vectors. As there are 5 possible agent or patient components, 3 possible action components and 3 possible questions, 8 input, 8 output and 3 question units were used.

- Random generation of 20 training examples from a training example distribution. A number of different training distributions were used (see Table 5.2), where the overlap between objects that appeared in the agent and patient positions varied from 0 (no noun appeared in both positions) to 5 (every noun appeared in both positions). In each case, with the exception of 0 and 5 noun overlap, noun(s) were left out of the patient position only. It is possible that the effect on generalization, by omitting patient position nouns only, is due to the limited number of patient components in the training set, rather than the limited degree of overlap between agent and patient positions. Thus, in the case of one, two and three noun overlap, a second training distribution was used (see Table 5.3), where nouns were left out of both positions, therefore ensuring that at least 3 different nouns appeared in the agent and patient positions.

- Random initialization of variable network weights from a uniform distribution between -1 and 1.
- Training of the network using the standard error backpropagation algorithm with 0.1 learning rate, no momentum term, and the sum of squares error function (Rumelhart et al., 1986) until outputs were within 0.4 of their target activation for all *output* units on all training patterns. Training was terminated after 50000 epochs if this criterion was not met. Each word or question was presented to the network one per time step, with the *context* and *tensor* units being reset to zero at the beginning of each sequence. Weights were updated at the end of each sentence-question pair (i.e., every four patterns).
- Testing of the network on 100 test sequences randomly generated from a test example distribution. The testing distribution associated with each training distribution is given in Tables 5.2 and 5.3. Importantly, since generalization across position is being tested, the training and testing distributions were not the same[5] (i.e., some nouns should not appear in some positions in the training set, but should appear in those positions in the test set). Two testing

[5]Except in the case where the training example distribution had an overlap of 5. This case, although not considered in the evaluation of strong systematicity, and was included for completeness.

Table 5.2: Training and testing example distributions where nouns were omitted from the patient position in the case where there was between 1 and 4 nouns appearing in both positions. The 0 (no noun appearing in both positions) and 5 (every noun appearing in both positions) overlap cases were included for completeness. In the 0 overlap case, nouns were omitted from both positions. The table shows only agent and patient objects (i.e., *B - Bill*; *J - John*; *K - Karen*; *M - Mark*; and *V - Vivian*). For the training sets, agent and patient objects were combined with every action and question. For the testing set, the superscript q indicates the set of objects that were queried, and therefore tested. The symbol $\times$ is the cartesian product operator generating all pair-wise combinations from elements in the respective sets.

Overlap	Train (agent,patient)	Test (agent,patient)
0	$\{B,J,K\} \times \{M,V\}$	$\{M,V\}^q \times \{B,J,K\}^q$
1	$\{B,J,K,M,V\} \times \{B\}$	$\{B,J,K,M,V\} \times \{J,K,M,V\}^q$
2	$\{B,J,K,M,V\} \times \{B,J\}$	$\{B,J,K,M,V\} \times \{K,M,V\}^q$
3	$\{B,J,K,M,V\} \times \{B,J,K\}$	$\{B,J,K,M,V\} \times \{M,V\}^q$
4	$\{B,J,K,M,V\} \times \{B,J,K,M\}$	$\{B,J,K,M,V\} \times \{V\}^q$
5	$\{B,J,K,M,V\} \times \{B,J,K,M,V\}$	$\{B,J,K,M,V\}^q \times \{B,J,K,M,V\}^q$

criteria were used. A response to a test sequence was considered correct when: (1) the maximally activated *output* unit had a target activation of one - maximum criterion; or (2) all *output* units were within 0.5 of their target activation - 0.5 criterion. During the test phase performance on the auto-association of input was not considered. Its purpose was to encourage the formation of internal representations during training, as discussed in the previous section.

- Each train-test trial was repeated 5 times, with modifiable weights randomly initialized at the beginning of each trial.

5.5.3 Results

In the case of the training and testing distributions from Table 5.2, the percentage of correct responses (maximum criterion) to the question vector on the test set averaged over 5 trials is given in Figure 5.4, where the bottom dashed line indicates

Table 5.3: Training and testing example distributions for the 1, 2 and 3 overlap cases, where nouns were omitted from both positions. The table shows only agent and patient objects (i.e., *B - Bill*; *J - John; K - Karen*; *M - Mark*; and *V - Vivian*). For the training sets, agent and patient objects were combined with every action and question. For the testing set, the superscript q indicates the objects that were queried, and therefore tested. The symbol $\times$ is the cartesian product operator generating all pair-wise combinations from elements in the respective sets.

Overlap	Train (agent,patient)	Test (agent,patient)
1	$\{B, J, K\} \times \{K, M, V\}$	$\{B, J, K, M^q, V^q\} \times \{B^q, J^q, K, M, V\}$
2	$\{B, J, K, M\} \times \{K, M, V\}$	$\{B, J, K, M, V^q\} \times \{B^q, J^q, K, M, V\}$
3	$\{B, J, K, M\} \times \{J, K, M, V\}$	$\{B, J, K, M, V^q\} \times \{B^q, J, K, M, V\}$

chance level response. The graph shows results only for noun-position combinations that did not occur in the training set. For example, when no noun appeared in both agent and patient positions in the training set, the network achieved a mean accuracy of 74%. When three or more of the possible five nouns appear in both positions in the training set, the network was 100% accurate on the test set.

For the same distributions, the percentage of correct responses using the 0.5 testing criterion is given in Figure 5.5. Note that chance level performance for this testing criterion is $\frac{1}{2^5} \times 100\%$ (= 3%), as there is a 50% chance of the activation of each *output* unit being on the "right" side of 0.5. For this criterion the network's mean accuracy was 64% with no overlap and 100% for three of more overlapping objects.

In the case of the training and testing distributions from Table 5.3, the percentage of correct responses using the maximum testing criterion, and the 0.5 testing criterion are given in Figures 5.6 and 5.7, respectively.

All trials except one converged to the training criterion within 50000 epochs. The single non-convergent case, occurred with the zero overlap training distribution. Training times for distributions given in Tables 5.2 and 5.3 are shown in Figures 5.8 and 5.9, respectively.

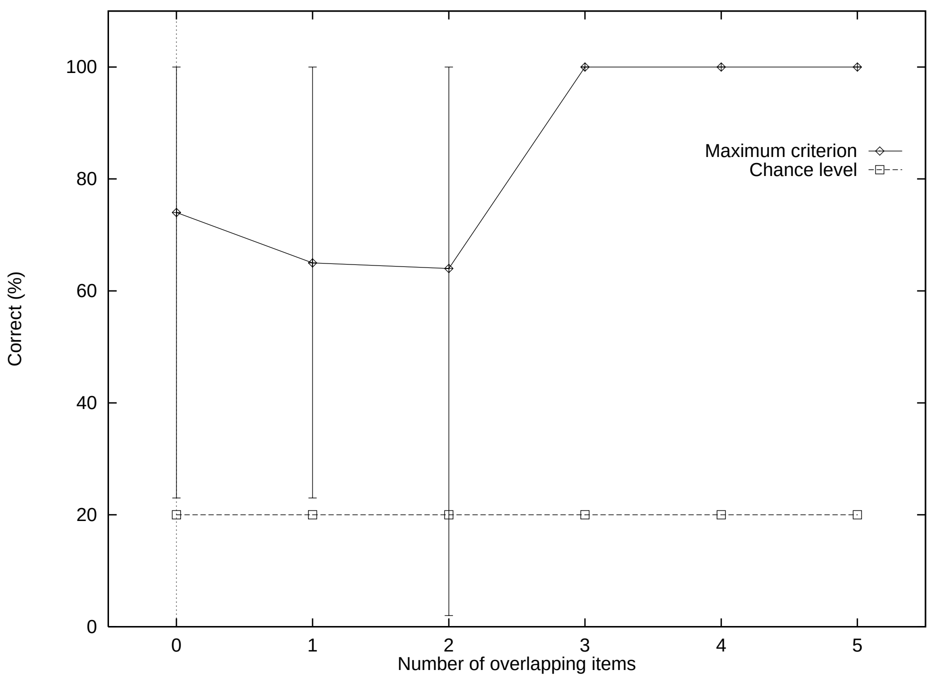


Figure 5.4: Generalization (maximum criterion) as a function of the number of overlapping items where items were omitted from the patient position in the 1 to 4 overlap cases (see Table 5.2). The percentage correct on the test set was averaged over 5 trials. The dashed line indicates chance level performance. Error bars indicate 95% confidence intervals.

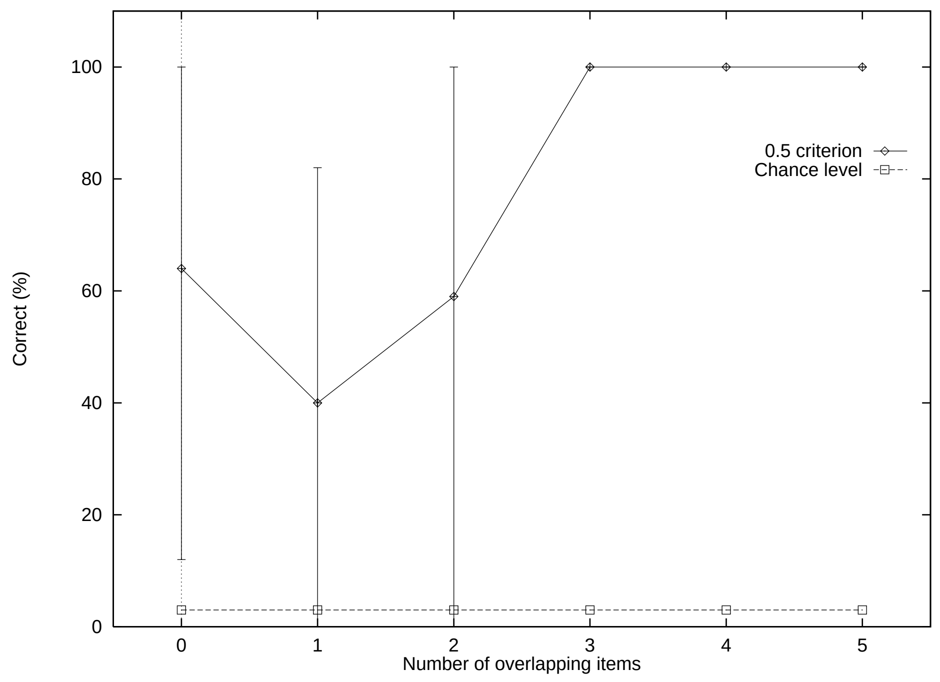


Figure 5.5: Generalization (0.5 criterion) as a function of the number of overlapping items where items were omitted from the patient position in the 1 to 4 overlap cases (see Table 5.2). The percentage correct on the test set was averaged over 5 trials. The dashed line indicates chance level performance. Error bars indicate 95% confidence intervals.

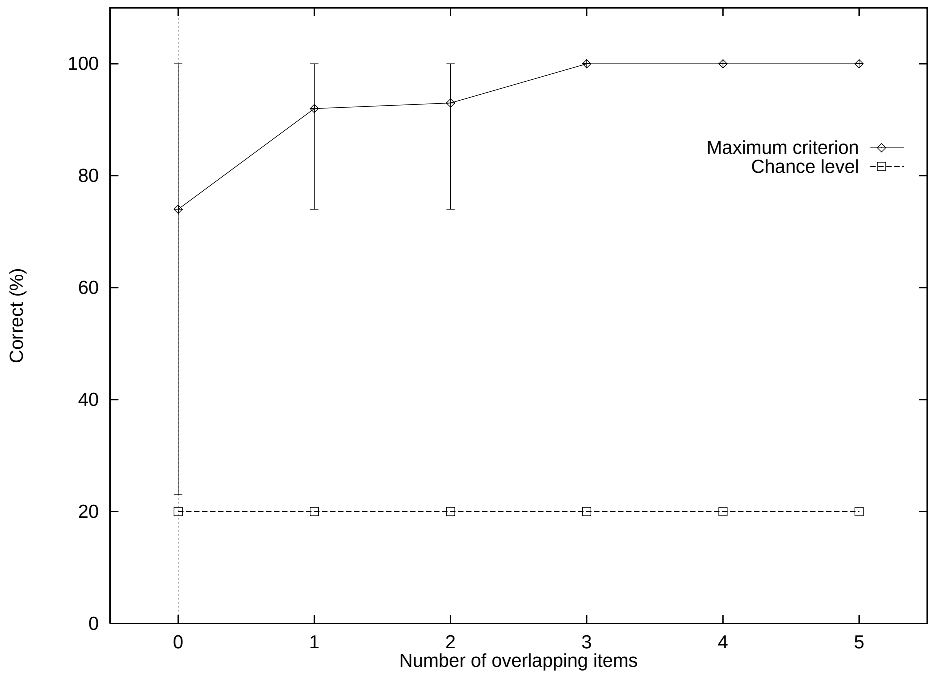


Figure 5.6: Generalization (maximum criterion) as a function of the number of overlapping items where items were omitted from both positions in the 1 to 3 overlap cases (see Table 5.3). The percentage correct on the test set was averaged over 5 trials. The dashed line indicates chance level performance. Error bars indicate 95% confidence intervals.

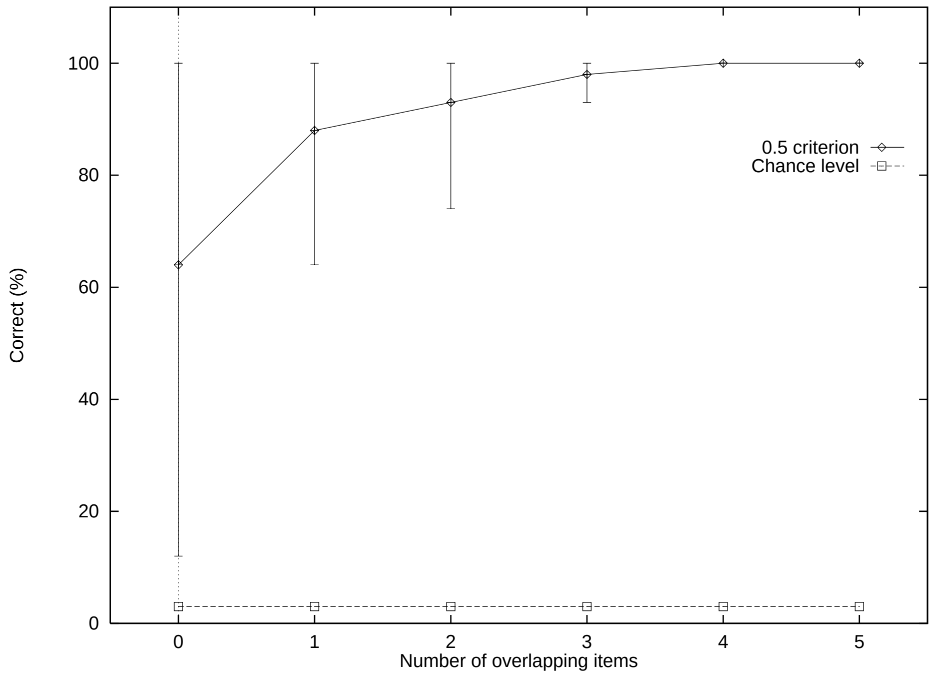


Figure 5.7: Generalization (0.5 criterion) as a function of the number of overlapping items where items were omitted from both positions in the 1 to 3 overlap cases (see Table 5.3). The percentage correct on the test set was averaged over 5 trials. The dashed line indicates chance level performance. Error bars indicate 95% confidence intervals.

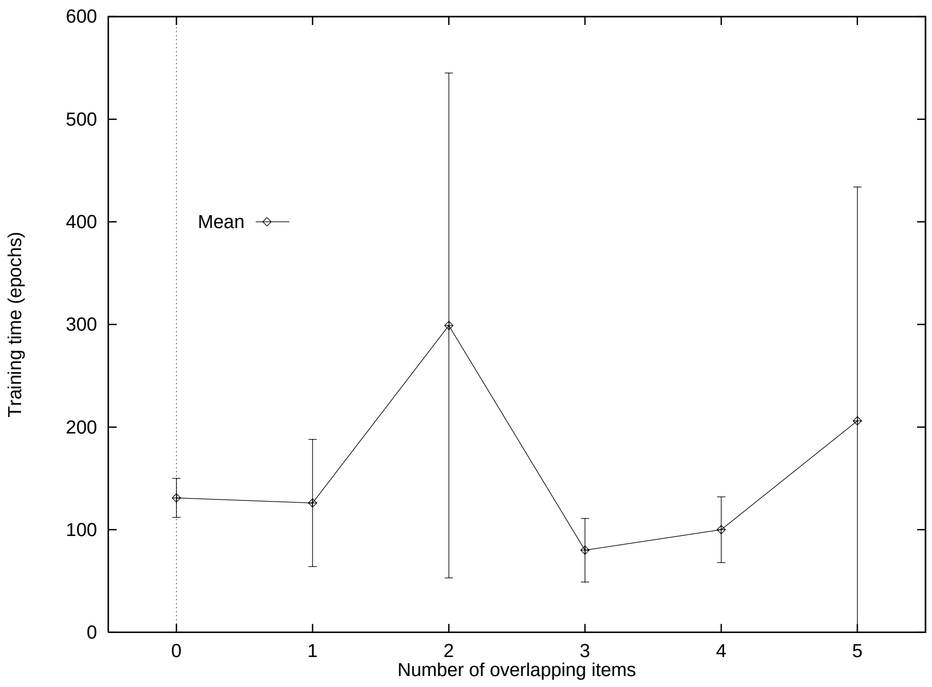


Figure 5.8: Mean training times as a function of overlap where items were omitted from the patient position in the 1 to 4 overlap cases (see Table 5.2). The zero overlap case, does not include the single non-convergent trial. Error bars indicate one standard deviation.

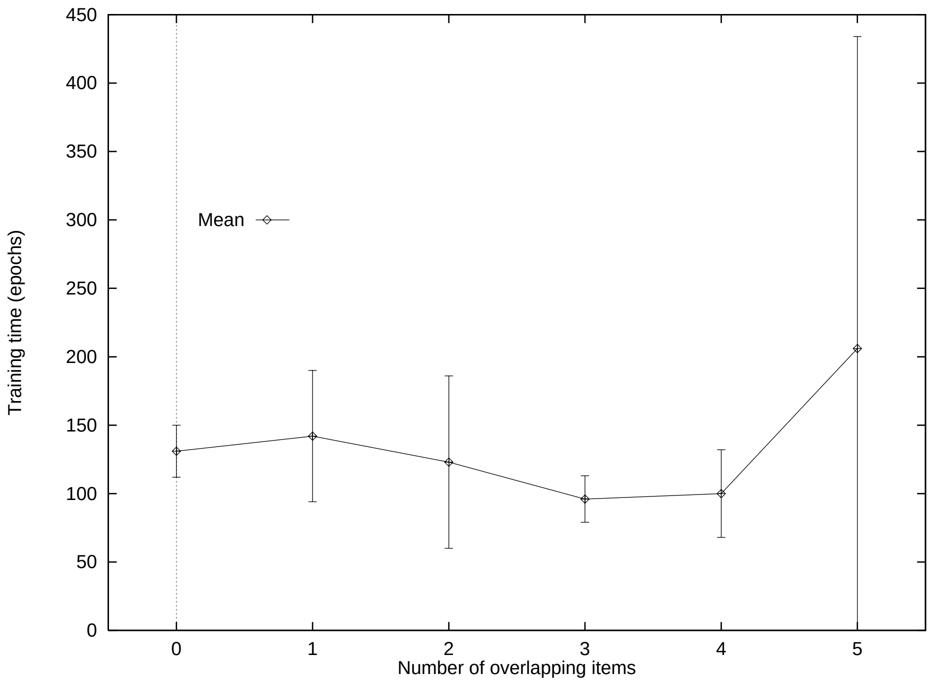


Figure 5.9: Mean training times as a function of overlap where items were omitted from both positions in the 1 to 3 overlap cases (see Table 5.3). The zero overlap case, does not include the single non-convergent trial. Error bars indicate one standard deviation.

5.5.4 Discussion and analysis

The point of these simulations was to test whether the tensor-recurrent network exhibits generalization across position above chance level with respect to this task (i.e., strong systematicity of inference). For the training distributions used in Table 5.2, the network showed perfect generalization to novel object-positions on all trials where there was an overlap of three or more nouns in both positions. Perfect generalization occurred for maximum and 0.5 testing criteria. Since the degree of generalization was consistently above chance level, the network is said to have exhibited strong systematicity of inference with respect to this task.

The strong systematicity exhibited by the network was a consequence of three factors: (1) architecture; (2) learning; and (3) environment (i.e., training set distribution). Each of these factors are discussed in turn.

Architecture

Strong systematicity was a consequence of separating the responsibility for representing component and position information. Partly, this separation was due to the architecture whereby component information was represented (independent of its position) at the *hidden* units, and position information was represented (independent of its component value) at the *role* units. The component-independence of position information was encouraged by the *bottleneck* units which attenuated the effect of the current input on the state.

Learning

The separation of component and position information, however, was also a result of the network's learning dynamics. It was assumed that backpropagation of error would orthogonalize the role vectors (which is crucial for the correct extraction of components). During one trial the activation vectors for the *role* units were recorded while training on a data set generated from a distribution where three nouns appeared in both agent and patient positions (see Table 5.2). Table 5.4

Table 5.4: Orthogonality between agent, action and patient role vectors measured as one minus the magnitude of the normalized dot product. Role vector $\vec{R}'_i = \vec{R}_i + \vec{C}_{bias}$, where $\vec{R}_i$ is the activity vector at the *role* units, and $\vec{C}_{bias}$ is the activity vector at the *cue* units resulting from the bias terms.

Weight updates	$\vec{R}'_{agent}, \vec{R}'_{action}$	$\vec{R}'_{agent}, \vec{R}'_{patient}$	$\vec{R}'_{action}, \vec{R}'_{patient}$
0	0.1	0.1	0.0
1000	0.8	0.7	0.0
4900	0.8	0.8	0.9

shows how the role vectors[6] become progressively more orthogonal with training. (The measure being one minus the magnitude of the normalized dot product so that zero implies collinear and one implies orthogonal for non-zero vectors.) Before any training (i.e., at zero weight updates), the role vectors associated with the agent and action components were almost collinear, as indicated by the orthogonality measure of 0.1. However, after 1000 weight updates, the two role vectors became more orthogonal, as indicated by an orthogonality measure of 0.8. At this stage of training, the action and patient role vectors were still collinear (as indicated by the orthogonality measure of 0.0). However, after 4900 weight updates, the two role vectors orthogonalized (with an orthogonality measure of 0.9). It is not necessary that the role vectors be perfectly orthogonal since the non-linear activation function at the output layer is capable of masking out residual vectors. This table demonstrates that before training the network was not systematic. The collinearity of the action and patient role vectors, for example, means that the network could not have extracted the associated verb without extracting the noun in the patient position. Therefore, the strong systematicity exhibited by the network was also a consequence of learning.

The network was also required to learn the appropriate cue vectors so that component representations could be extracted from the tensor representation. During the same trial, the activation vectors at the *cue* units were also recorded. Table 5.5

[6]Where the role vector $\vec{R}'_i = \vec{R}_i + \vec{C}_{bias}$.

Table 5.5: Collinearity between cue (at time step 4) and role vectors concerned with the extraction and construction (respectively) of the agent, action and patient positions measured as the magnitude of the normalized dot product. Role vector $\vec{R}_i' = \vec{R}_i + \vec{C}_{bias}$, where $\vec{R}_i$ is the activity vector at the *role* units and $\vec{C}_{bias}$ is the activity vector at the *cue* units resulting from the bias terms. At time step 4, cue vector $\vec{C}_i' = \vec{C}_i + \vec{R}_4$, where $\vec{C}_i$ is the activity vector at the *cue* units, and $\vec{R}_4$ is the activity vector at the *role* units at time step 4.

Weight updates	$\vec{C}'_{agent}, \vec{R}'_{agent}$	$\vec{C}'_{action}, \vec{R}'_{action}$	$\vec{C}'_{patient}, \vec{R}'_{patient}$
0	0.8	0.9	0.9
1000	0.1	1.0	1.0
4900	0.8	0.9	1.0

Table 5.6: Collinearity before training between cue (at time step 4) and role vectors concerned with the extraction and construction (respectively) of the agent, action and patient positions measured as the magnitude of the normalized dot product. Role vector $\vec{R}_i' = \vec{R}_i + \vec{C}_{bias}$, where $\vec{R}_i$ is the activity vector at the *role* units and $\vec{C}_{bias}$ is the activity vector at the *cue* units resulting from the bias terms. At time step 4, cue vector $\vec{C}_i' = \vec{C}_i + \vec{R}_4$, where $\vec{C}_i$ is the activity vector at the *cue* units, and $\vec{R}_4$ is the activity vector at the *role* units at time step 4.

Weight updates	$\vec{R}'_{agent}$	$\vec{R}'_{action}$	$\vec{R}'_{patient}$
$\vec{C}'_{agent}$	0.8	0.9	1.0
$\vec{C}'_{action}$	0.8	0.9	0.8
$\vec{C}'_{patient}$	0.6	0.8	0.9

shows how the cue vectors[7] line up with the appropriate role vectors with training. (The measure being the magnitude of the normalized dot product.) For example, after 4900 weight updates, the cue vector for extracting the patient component was collinear with the role to which the patient component was composed (indicated by the collinearity measure of 1.0). Before training, cue vectors were collinear with their associated role vectors. However, the cue vectors were also collinear with the other role vectors as indicated in Table 5.6. Thus, before training it was not possible to extract the target component without extracting other components.

[7]Where, at time step 4, the cue vector $\vec{C}_i' = \vec{C}_i + \vec{R}_4$.

Environment

The third factor influencing the demonstration of strong systematicity of inference was the environment in which the training examples were generated (i.e., the training example distribution). With the training distributions given in Table 5.2, perfect generalization across position was exhibited on all trials when there was an overlap of three or more nouns in the agent and patient positions. However, when the number of overlapping items was reduced to two nouns, generalization across position dropped sharply. In the two item overlap case, the network no longer exhibited above chance level performance with greater than 95% confidence for both maximum (Figure 5.4) and 0.5 (Figure 5.5) testing criteria. Performance in the one item overlap case was similar for the maximum testing criterion (Figure 5.4), and worse for the 0.5 testing criterion (Figure 5.5).

In the one and two item overlap cases, there were only one and two objects (respectively) appearing in the patient position in the training set. Correct extraction of components requires the network to learn to generate a common role vector for the patient[8] position. With only one or two objects in the patient position, the network did not generalize to the case whereby a common role vector was generated for the remaining objects in the patient position. Since only one cue vector is generated per question, it becomes increasingly difficult to maintain similarity between the single cue vector and multiple role vectors generated for the same position. Consequently, performance decreased in these cases.

In the one, two and three item overlap cases, training distributions were changed so that at least three objects appeared in both agent and patient positions while the same degree of overlap was maintained (see Table 5.3). For example, in the one item overlap case, *Bill*, *John* and *Karen* appeared in the agent position and *Karen*, *Mark* and *Vivian* appeared in the patient position. With these training distributions, test performance improved to the point where, with at least 95% confidence, the network generalized to novel object-positions with an accuracy of

[8]The network must also learn to generate a common role vector for the agent and action positions.

at least 74% (maximum testing criterion, see Figure 5.6), and at least 64% (0.5 testing criterion, see Figure 5.7). Performance was also higher in the two overlap case.

The number of network weights also influences the capacity to generate common role vectors for each position. The *bottleneck* units were introduced to attenuate the effect of the input on the generation of state, and therefore role vectors. Increasing the number of *bottleneck* units increases the degrees of freedom, and therefore the sensitivity of the role vectors to the current input. In other words, it is more likely that the generated role vectors are specific to component, rather than position information. Therefore, an increase in these weights, decreases the degree of generalization across position.

The network also demonstrated above chance level performance, with greater than 95% confidence, for both maximum and 0.5 testing criteria when there was no overlap across agent and patient positions. This result means that the network will also demonstrate generalization from agent to action, and from action to agent or patient positions. Thus, it raises the question of whether the network has too strong an architectural bias. Should generalization across position occur when there is no overlap. For example, Hadley cites the case where children generated new verbs from previously learnt nouns (e.g., *It was bandaided*). Clearly, the degree to which people generalize across position is an empirical question, and one not addressed in Hadley's definition of systematicity. Hadley's concern, and the concern of this thesis, has been to establish some above chance level degree of generalization across position. However, in the next chapter, architectural properties that permit generalization across position *only* in the case of overlap are discussed.

5.6 Summary and conclusions

In the previous chapter, it was shown that a common property preventing a demonstration of strong systematicity of inference in the Connectionist models examined was an independence between access weights (i.e., the weights that implement the functions that access component representations from the various positions within a complex representation). It was suggested that a dependency between these weights would be sufficient to demonstrate strong systematicity of inference. The purpose of this chapter was to implement and test such a dependency in the form of the tensor-recurrent network.

Weight dependency was implemented in the tensor-recurrent network by accessing component representations via the dot product operator in tensor calculus. In this way, the multiple subspaces provided by tensor units, which hold representations of complex objects, can be collapsed down to a single subspace from which component representations may be extracted independent of their position. The dependency across access weights was achieved as access was, ultimately, via the same set of weights. The use of tensors assumes appropriate component and role representations (for the construction of tensor representations), and cue representations (for the extraction component representations). Simulations showed how these representations were learnt in the tensor-recurrent network by back-propagating error signals through the *tensor* units to the weights that generate the component, role and cue vectors. Successful extraction of component representations also relies on the network generating common (i.e., item independent) role vectors for each position within a complex object. Simulation results showed that item independent role vectors were learnt when sufficient objects appeared in each position in the training set. The number of weights implementing the function that generates role vectors influences the degree of generalization across position. Thus, from the simulations it was concluded that the architectural biases assumed in the tensor recurrent network, which were, together, sufficient for strong systematicity of inference on the querying of 3-tuples task were:

- *tensor* units, which construct representations of complex objects as the outer product of internal component and role representations;

- *inner product* units, which collapse the multiple representational subspaces down to a single subspace from which component representations were extracted;

- backpropagation learning dynamic, which allows the appropriate internal component, role and cue representations to be learnt; and
- component-independent, position-dependent generation of role vectors.

The tensor-recurrent network was designed specifically to address the issue of strong systematicity of inference. It may be that the network is too systematic, in that it generalizes to cases not supported by empirical evidence. This possibility was suggested when the network demonstrated some degree of generalization across position in the case where there was no overlap across agent and patient positions in the training set. In the next chapter, the extent to which the assumptions made in the tensor-recurrent network can be relaxed are discussed. In addition, the significance of the work in this thesis is discussed in terms of approaches to cognitive modeling and the relationship of this work to the Classical paradigm.

Chapter 6

Discussion

6.1 Introduction

The purpose of this chapter is to reflect on the work presented in the previous chapters. Specifically, a discussion is given in relation to: the implications of these results for Connectionist (and other) approaches to cognitive modeling (section 2); a limitation of the tensor-recurrent network solution, and a possible extension which may address the problem (section 3); and the relationship of the Connectionist approach presented here to the Classical paradigm (section 4). A summary is provided in section 5.

6.2 Implications

The problem that systematicity posed for the Connectionist approach to cognition was an explanation as to how the acquisition of systematic behaviour could be a necessary property of a Connectionist architecture. That is, in terms of a learning framework, how can learning to represent and process some instances of a structured object necessarily generalize to other instances conforming to the same structure? In this section, the implications arising from the results obtained in addressing this question are discussed.

6.2.1 Same training and testing distribution assumption

In chapter 3, the problem of the necessary acquisition of systematic behaviour was framed in terms of probably approximately correct (PAC)-learnability. That is, a Connectionist model was considered to have necessarily acquired systematic behaviour over some suitably structured domain when a high degree of generalization over that domain was obtained, with a high degree of confidence (i.e., over repeated trials) with at most a polynomial amount of computational resource in some parameter that measures the size of the behaviour being learnt. Using this criterion, the feedforward network was shown to demonstrate systematicity with respect to the auto-association of N-tuples task.

Although the feedforward network demonstrated systematicity as defined in terms of PAC-learnability it was suggested that this definition was too weak for the purposes of cognitive models. For although this definition is suitable in terms of computational feasibility (i.e., any model requiring more than polynomial resource is impractical as there is rarely enough resource available to learn even moderately sized behaviours), it did not consider the amount of resource actually required. Consequently, a model may be computationally feasible, but still require more resource than is ever needed by people. Thus, an alternative definition of systematicity was considered, which was Hadley's strong systematicity, whereby a model is said to exhibit the acquisition of systematicity if it shows above chance level generalization to component-position combinations that it had never been trained (i.e., generalization across position). In chapter 4, the same feedforward network was examined on the same task for strong systematicity. It was found that the feedforward network could not exhibit strong systematicity. The implication of this result is concerned with an assumption regarding the training and testing distributions.

The PAC-learnability framework generally assumes that the examples on which the network was trained are drawn from the same distribution as the examples used for testing generalization[1]. However, a test for strong systematicity, by definition,

[1]Although, Bartlett (1992) has considered the case where the training-testing distribution

requires the training and testing distributions to be different. More specifically, if a model is to demonstrate strong systematicity then there must be objects from the domain that are drawn from a training distribution P_{train}, such that $P_{train}(R_i = x) = 0$ (i.e., the probability of object x appearing in role/position R_i is zero), but are drawn from a test distribution P_{test}, such that $P_{test}(R_i = x) \neq 0$ (i.e., the probability of the same object (x) appearing in same role/position R_i is not zero). Thus, the implication for Connectionist (and other learning) approaches to cognitive modeling is that criteria derived from the PAC-learning framework, where there is an assumption that the training and testing distributions are the same, are potentially weak criteria in that they may admit cognitive models that cannot account for particular behaviours, as was the case, for example, with the three-layer feedforward network with respect to strong systematicity[2].

6.2.2 Structured networks

Traditionally, the approach taken in Connectionism has been to specify general-purpose, computationally-sufficient[3] architectures; and then provide a learning dynamic, which allows for the acquisition of some interesting behaviour. Numerous results already exist regarding the capacity of Connectionist networks to compute arbitrary functions under a variety of basis functions. For example, it has been shown that feedforward networks with sufficient sigmoidal (Funahashi, 1989), squashing (Hornik et al., 1989), and gaussian (Hartman, Keeler, & Kowalski, 1990) units are universal approximators. Furthermore, Sato et al. (1990) presented a recurrent network as a universal approximator to a general class of dynamical systems, and Siegelman and Sontag (1991) proved that a recurrent network with finite sigmoidal units is computationally equivalent to a universal Turing machine. The implication of the negative results of the three-layer feedforward and recurrent networks of chapter 4 is that it is not sufficient to specify computationally-general mechanisms,

changes by a small constant.

[2]Amsterdam (1988) has criticized the formal learning approach, from a philosophical perspective, on the basis that it only considers learning from naturally occurring examples, and therefore does not consider the possibility of acquiring concepts that do not occur naturally.

[3]In the sense, that with sufficient internal units, a network can compute an arbitrary function.

one must also consider behaviour-specific biases. For example, although the feed-forward network in chapters 3 and 4 has sufficient resources to represent (and even learn) behaviours over structured domains, it requires too many examples to be an adequate model in regard to the necessary acquisition of systematic behaviour.

Of course, it has generally been recognized that Connectionist learning models require some form of architectural bias for the acquisition of complex behaviours. The important question has been, and still is: what sort of biases are necessary, or even sufficient for the acquisition of these behaviours? In chapter 4, it was shown that the property preventing the networks from exhibiting strong systematicity was the independence between the weights that access component representations from the various roles within a complex representation. In chapter 5, weight dependence was introduced into the tensor-recurrent network. The characteristic difference between the tensor-recurrent network and the networks in chapter 4 was that components were accessed via a common representational subspace. Thus, learning to access a component object representations from one position generalized to other positions. This characteristic property was also evident in the simple recurrent network in a demonstration of strong systematicity of representation.

The architectural organization of the tensor-recurrent network was sufficient to demonstrate strong systematicity on an inference task. However, it was suggested that the network was too strongly biased in that it exhibited generalization not apparent in the training data (i.e., when there was no overlap). The implication of this result is that the network may not exhibit appropriate behaviour where generalization across position is not required. In the next section, this limitation is discussed along with a possible extension to address this limitation.

6.3 Limitation and possible extension

In the three-layer networks analyzed in chapter 4, components were extracted from complex representations, held at the *hidden* units, via a single layer of weighted connections linking the *hidden* and *output* units. Figure 6.1 characterizes the extraction process for such networks, where the dashed arrows indicate modifiable

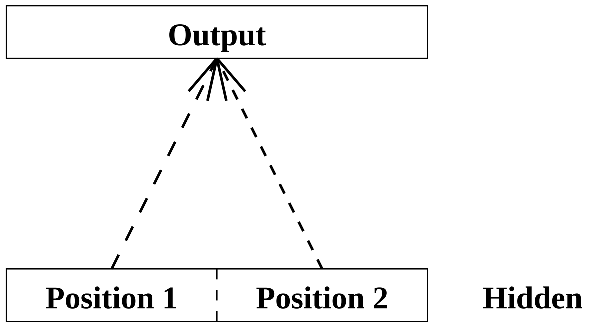


Figure 6.1: Characterization of the extraction of component representations from positions 1 and 2 for the three-layer networks in chapter 4 (i.e., with one hidden layer). Dashed arrows indicate modifiable-weighted connections. The diagram shows only the hidden and output layers.

connections. In the tensor-recurrent network, component representations were extracted via an intermediate layer (the *inner product* units), which extracted the desired component subspace from the *tensor* units. Figure 6.2 characterizes the extraction process for the tensor-recurrent network, where solid arrows indicate fixed-weighted connections. The tensor-recurrent network assumed, as one of its architectural biases, unit connectivity for implementing the inner product. Thus, the process of collapsing subspaces down to a single subspace was already provided, regardless of the degree of overlap evident in the training set. Therefore, the network was able to exhibit some degree of generalization across position in the case where there was no overlap.

One alternative is to make the weighted connections between the two internal layers of the tensor-recurrent network learnable. In this way, there is the potential for the network to learn weights which collapse subspaces only in the case where there was a sufficient degree of overlap in the training set.

The first point to consider is whether a network with modifiable weights between two hidden layers can demonstrate strong systematicity of inference. Suppose a network has a similar internal organization to that of the tensor-recurrent network, except that the fixed weights between the two internal layers are replaced with modifiable weights (Figure 6.3). Suppose also that the resources of this network are such that there are two units per position in the first internal layer and two units in the second layer, and that activation functions of the second internal layer and

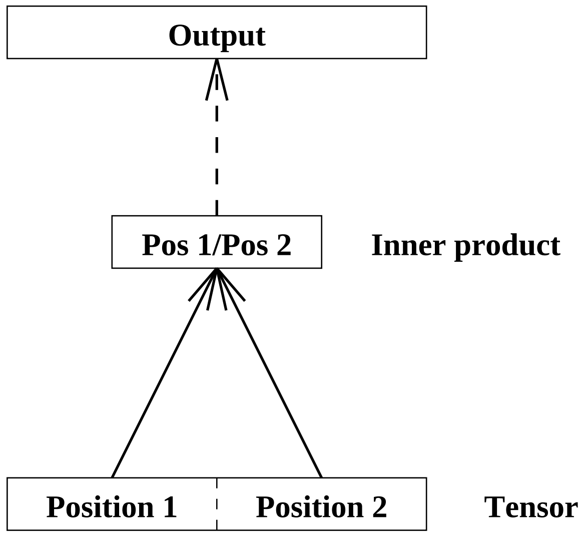


Figure 6.2: Characterization of the extraction of component representations from positions 1 and 2 for the tensor-recurrent network. Dashed arrows indicate modifiable-weighted connections. Solid arrows indicate fixed-weighted connections. The diagram shows only the two hidden layers (i.e., tensor and inner product) and the output layer.

output units are the identity and sigmoidal functions, respectively. Thus, two units are sufficient to represent all possible components in a given position (Kruglyak, 1990). The network must learn the weights between the three layers of units to demonstrate strong systematicity of inference on the querying of 2-tuples task.

Assume also that the network has learnt to buffer the input onto the two subspaces so that the relative positions within each of the subspaces is the same. Then the question is: Can the network learn the extraction part of the inference task without having to be trained on all components in both positions?

Suppose, for example, there are six possible components in each position, and the network is trained on examples where all components appeared in the first position, but not in the second position. After training, the network has learnt the mapping depicted in Figure 6.4. The diagram assumes that the network has also learnt to select, depending on the query, between the two subspaces by the time they are represented at the first hidden layer (i.e., only one of the two components is active at the first hidden layer). The question is: how many components in the second position are required to learn the same mapping from the second subspace as was learnt from the first subspace.

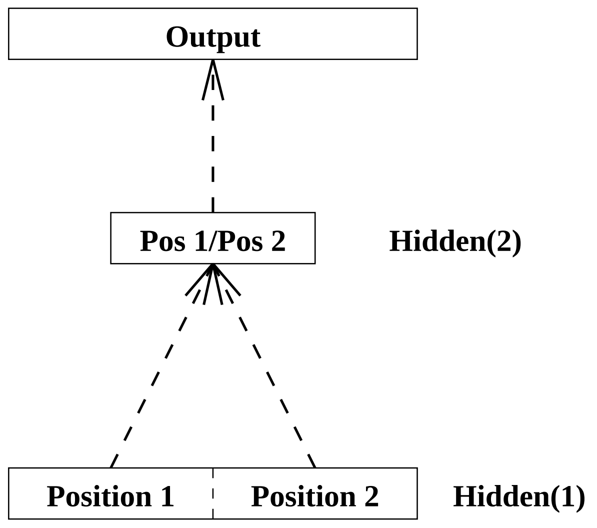


Figure 6.3: Characterization of the extraction of component representations from positions 1 and 2 for a network with two hidden layers linked by modifiable-weighted connections. Dashed arrows indicate modifiable-weighted connections. The diagram shows only the two hidden layers and the output layer.

The answer is at most three components. The weights from the first to the second hidden layer perform a linear transformation. In the case where the network has been trained on no more than two components in the second position there are an infinite number of weight vectors that satisfy the mapping specified by the training set (Figure 6.5(a)). In the case where there were three components, there is only one solution (Figure 6.5(a)), and therefore the network generalizes to the other three positions. By sharing a common representational subspace at the second layer, the network has already learnt the mapping from the second hidden layer to the output layer for all components since all components appeared in the first position and were mapped through the same subspace. Thus, the network has the possibility of exhibiting strong systematicity, given also the buffering and querying assumptions. The important condition is that the dimensionality of the positional and common subspaces is less than the number of possible components. If there are as many dimensions as components then some dimensions would only be trained on one type of point, and therefore there would not be enough information in the training set to uniquely specify the mapping. For this reason, the three-layer networks in chapter 4 could not demonstrate strong systematicity, since the

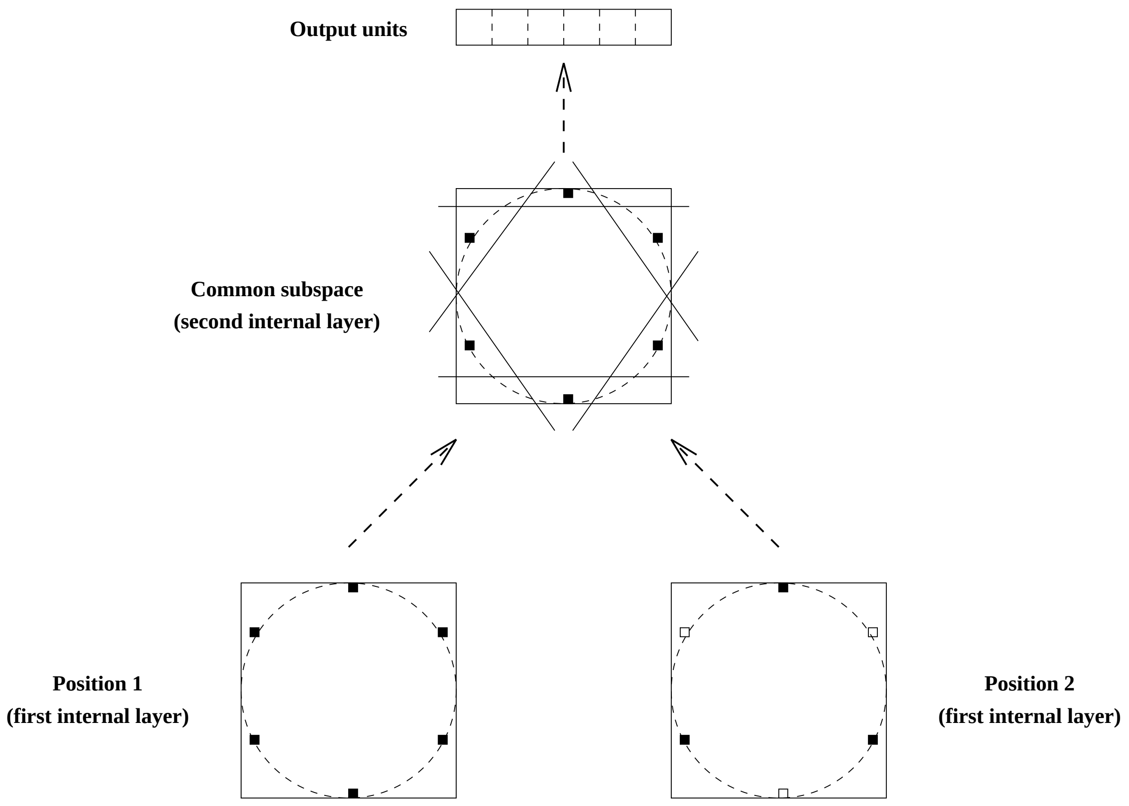


Figure 6.4: An example of the internal organization of a network, with two units per subspace in the first internal layer and two units for the common subspace in the second internal layer, learning the querying of 2-tuples task. Dashed arrows indicate modifiable-weighted connections. Solid squares indicate training points and empty squares indicate test points. Lines through the common subspace indicate characteristic positioning of hyperplanes for the correct extraction of component representations by the output units. Dashed circle indicates the characteristic positioning of training points for an arbitrary number of components per position.

number of output units was the number of components (which was a condition on the external representations of objects since there should not be any *a priori* similarity between external object representations).

There are a number of ways in which the dimensionality of component subspaces could be enforced, or at least encouraged. One way is to limit the number of units in the second hidden layer. The difficulty with this approach is that it is generally more difficult to learn a function with fewer weights as the solution space is smaller and there are fewer trajectories which lead to a solution. Alternatively, excessive weights can be provided initially, and subsequently removed during the course of training where they have shown to be irrelevant. Weights can be effectively removed

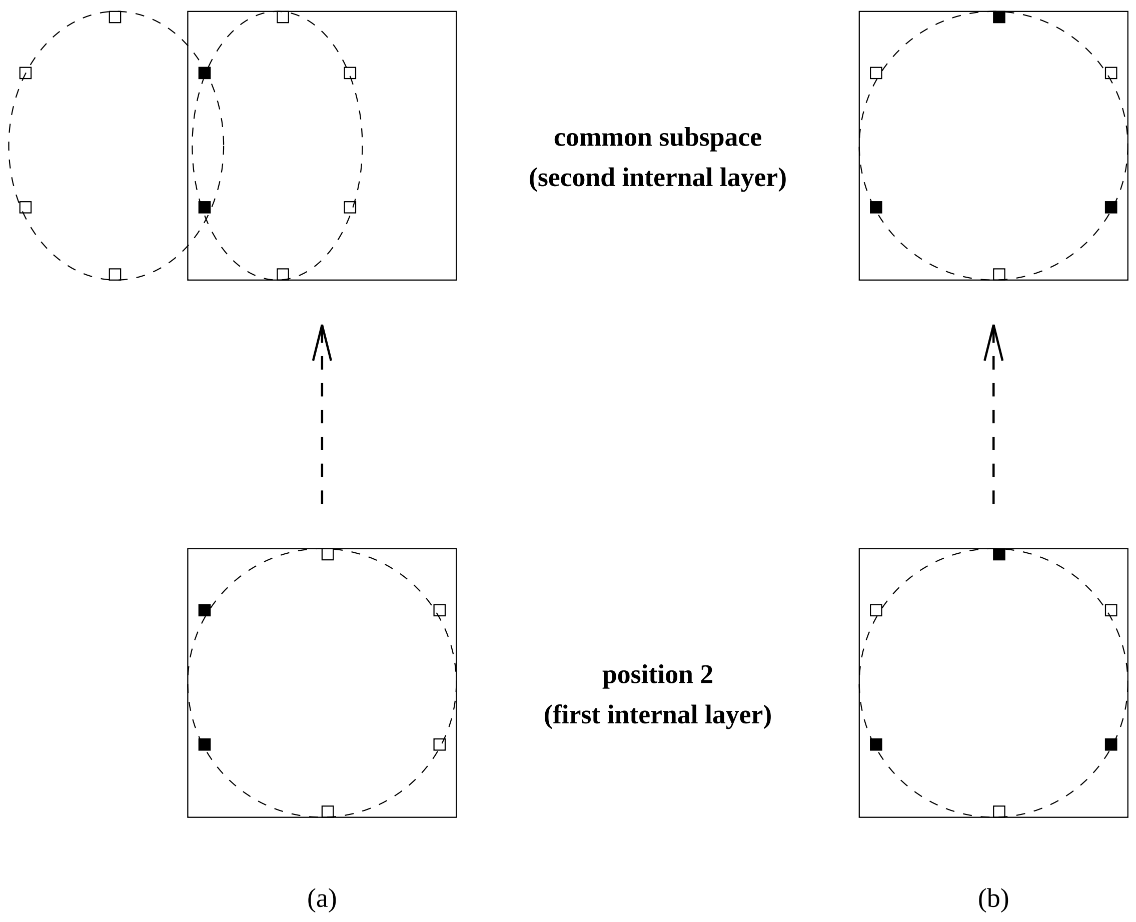


Figure 6.5: Number of components in the second position required in the training set to demonstrate strong systematicity. With only two components (a) there is not enough information to constrain the possible mappings. Two possible solutions which satisfy the mapping specified by the training points (solid squares) are shown. The correct mapping can be uniquely described with three components (b). Empty squares indicate test points. Dashed arrows indicate modifiable-weighted connections which implement the mapping. Dashed circle indicates the characteristic positioning of training points for an arbitrary number of components per position.

with the use of an additional error term that penalizes unnecessary weights, as was shown, for example, in the work of Nowlan and Hinton (1991) and Zemel (1993). The error function not only encourages the network to find the target function, but to do so with the fewest necessary weights.

Thus, with respect to the querying of 2-tuples example, it is possible for the network with two hidden layers to be given an excessive number of units (i.e., greater than two in the second hidden layer), but make use of only a small portion of those units. In doing so, it is possible that the network learnt to collapse the subspaces from the previous layer to a single common subspace where the dimensionality is less than the number of possible components, and therefore demonstrate generalization across position.

Of course, it would require simulations to verify whether such additional terms in the error function would result in a demonstration of strong systematicity. However, error terms that reduce the number of weights could still result in solutions where all subspaces are collapsed to a common subspace, since this solution has the fewest number of weights (i.e., two hidden units in the second internal layer are sufficient to encode arbitrarily many components independent of their position).

What is required is to collapse component subspaces only in the cases where there is overlap evident in the training set. In the case where there is no overlap, the subspaces should remain separate. Figure 6.6 characterizes this situation for the *agent-action-patient* domain. In the early stages of training (Figure 6.6(a)), when there is no apparent or significant[4] degree of overlap between positions, component subspaces remain separate. This situation is likely to occur, for example, when the network is given excessive numbers of units so that there is no necessity for the dimensionality of each subspace to be less than the number of possible components. In the later stages of training, when overlap has been detected, the network reorganizes the weights associated with the overlapping subspaces so as to map them onto a common subspace (Figure 6.6(b)). In this way, generalization across position only occurs with components in the overlapping positions. Gen-

[4]One could consider a threshold, in which the degree of overlap must be exceeded before collapsing subspaces.

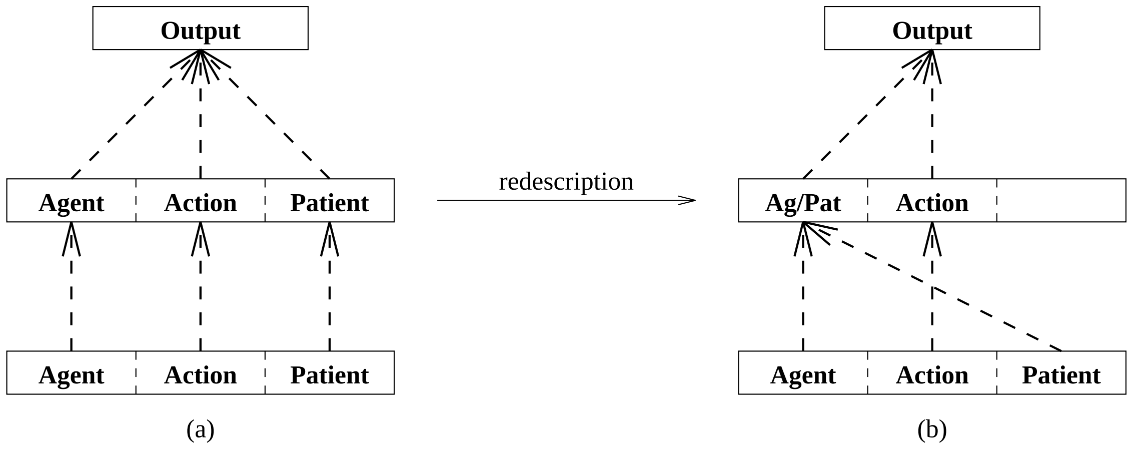


Figure 6.6: Process of redescribing internal representations. With three subspaces at the second internal layer (a), the network can perform the task, but without demonstrating strong systematicity. By collapsing the two overlapping subspaces (i.e., agent and patient) the network demonstrates strong systematicity, but only with respect to these two positions, not with respect to the action position (b). Dashed arrows indicate modifiable-weighted connections.

eralization to novel components in the action position would not occur since the network has never been trained to map the new component in the action position (in the second internal layer) to the output. Remembering that each output unit is dedicated to a single component (by the requirement that there is no *a priori* similarity between the external representations of components). Therefore, the weights from the action subspace (second layer) to the output unit have only been trained on one type of point (i.e., points without the new component).

The question, of course, is how could such training behaviour be a property of a Connectionist architecture? One possibility is to incorporate an objective function that encourages this behaviour. In designing an appropriate objective function, one needs to consider the cases in which representational subspaces should be brought together, and when they should not. Two subspaces, or internal representations within those subspaces should brought together when they map to the same output. For example, suppose vector representations $\vec{x}_1$ and $\vec{x}_2$ map to the same (or similar) output vector $\vec{y}$. Then, the weights which generated $\vec{x}_1$ and $\vec{x}_2$, should be changed so as to reduce the distance between those two vectors (i.e., collapses their representations down to a single vector representation). Note that in the standard objective functions, which only consider target and network output, the two vectors will not necessarily come together as a result of error backpropagation, because in an underconstrained network there are many points in the domain that map to the same point in the co-domain. So long as the common point ($\vec{y}$) in the co-domain is near the target, the two vectors ($\vec{x}_1$ and $\vec{x}_2$) will not necessarily be brought together through training. In the other case, where $\vec{x}_1$ and $\vec{x}_2$ do not map to the same output, training would proceed as usual (i.e., by reducing the difference between output and target vectors independent of each other).

The interesting point about such a process is its potential relationship to Karmiloff-Smith's (1992) *representation redescription hypothesis*. Karmiloff-Smith hypothesizes that cognitive development undergoes a number of phases (possibly concurrent) of which behavioural mastery is just one. During the acquisition of a competent level of performance on some task, additional processes are redescribing the behaviour into representations that are *explicit* relative to some other processes. This process of redescription allows for the transfer of ability on one task to other structurally related tasks. The transfer of ability is analogous to the way procedures and functions are reusable by other procedures and functions in programming languages. The process described above has a similar quality. The network in Figure 6.6(a) has the capacity to learn the target behaviour, but not in a strongly systematic way with its current organization of internal representations. However, if the representations are reorganized (redescribed) as in Figure 6.6(b), then the network has the capacity to demonstrate strong systematicity with respect to the agent and patient positions.

6.4 Relationship to the Classical paradigm

Having demonstrated a network that is strongly systematic, what then is its relationship to the Classical paradigm? In the Classical explanation of cognition, cognitive representations are symbols[5], or symbol structures (i.e., symbols that

[5]The symbols, by themselves, are abstract entities. The physical realization of a symbol may be for example, a **1 0 1 1** state of a register, or a 0.4mV activation potential of a neuron.

are composed of other more primitive symbols, for example aRb), and cognitive processes are sensitive to the structure of those representations. Since processing capacity is defined with respect to symbol structures, all objects conforming to that structure are processable. For example, a symbolic process called *first*, defined as: *first*: $(XY) \rightarrow X$ takes the structured symbol XY, composed of two symbols X and Y, and returns the symbol X. The symbol XY stand in some correspondance relationship to a complex object, which is external to the cognitive system, such that its component objects correspond to the more primitive symbols X and Y. The structural relationship between component symbols mirrors a structural relationship between component objects. Thus, the symbol XY may correspond to any external complex object with the same relationship between its component objects. The important point is that the computational processes of Classical architectures are defined at the level of symbols, therefore the capacities of these architectures come in clumps defined by the symbols on which they operate. So, systematicity is built into a Classical architecture.

In the tensor-recurrent network, external representations are mapped via weighted connections to internal subspaces. After the network has been trained to the point where it behaves systematically on the querying of 3-tuples task, the internal representations generated by the network are examples of symbolic representations. For instance, at the first time step, a similar role vector is generated regardless of the object that is presented to the network. The first role vector ($\vec{R}_1$) is a symbol denoting the first component object, and its physical realization is its *role* unit activations (e.g., **.1 .9 .5 .7**). At the same time, the external object is also mapped to a hidden unit vector, for example $\vec{H}_{John}$, which denotes in this case the object *John*. The physical realization of this symbol is its corresponding *hidden* unit activations. The role symbol is an example of a type (i.e., a symbol denoting a set of more primitive symbols), and the symbol (realized at the hidden units) is an example of a token (i.e., element from a set of symbols, or a type instance). The binding of the token to its type is performed by the outer product operator and represented at the *tensor* units. The construction of more complex symbols is achieved by adding the resulting tensors. Thus, the physical realization of the complex symbol "John loves Mary", is its corresponding *tensor* unit activations.

The similarity between a Classical architecture and the tensor-recurrent network would be of descriptive value only except that the symbols themselves are accessible by other processes within the network. Given an appropriate cue vector, a symbol is selected from a more complex symbol at the *tensor* units by the inner product operator, implemented by appropriate connectivity between *cue*, *tensor* and *inner product* units. It is because the constituent symbol can be extracted independently of the object to which it currently corresponds that the network exhibits systematicity.

It would appear that the tensor-recurrent network solution offers no more than a Connectionist implementation of a Classical architecture. However, there is one important difference, and that difference is concerned with the acquisition of systematic behaviour. Classical architectures, by definition, operate at the level of symbols and symbol structures. The grounding of symbols is assumed to occur by some perceptual level process that maps external stimuli to their corresponding symbols. Beyond that, cognitive behaviour is a process of symbol manipulation. Therefore, learning in a Classical architecture, like any other behaviour is a process of symbol manipulation. For example, learning the structure sensitive process *first* (see above) can only occur as a consequence of generating and testing possible symbolic mappings. Possible mappings are, $XY \rightarrow X$; $XY \rightarrow Y$; and $XY \rightarrow YX$, which the architecture could either accept or reject on the basis of environmental evidence. The problem with this form of learning is that the architecture can only progress from one systematic behaviour to another. At no point is there a transition from unsystematic to systematic behaviour. The lack of progression is because **the perceptual system on which a Classical architecture is grounded is not accessible to a Classical learning system**.

By contrast, in the tensor-recurrent network, the perceptual system (i.e., the weights that map external representations to symbols) is part of the learning process. Accessibility is granted in the tensor-recurrent network by linking external stimuli to their symbols via continuously differentiable functions. In this way, the relationship between external representations (stimuli) and their corresponding symbols can be learnt by a non-symbolic hill-climbing strategy, such as backpropagation. Therefore, it allows the tensor-recurrent network, as was demonstrated in chapter 5, to exhibit a transition from unsystematic to systematic behaviour.

Thus, the contribution of Connectionism is not in an alternative explanation for systematicity (i.e., the processing is still symbolic where it needs to be), but in an explanation of how the acquisition of systematic behaviour can be a necessary consequence of a network's internal organization and a non-symbolic learning process.

6.5 Summary

In this chapter, the implications of the results from this thesis were discussed. Two general lessons to be taken from this work in regard to cognitive modeling are: (1) The assumption that the training and testing distributions are the same may lead to criteria for cognitive architecture that are too strong. Therefore, models designed with this assumption are potentially incomplete, since for cognitive properties, at least with respect to strong systematicity, this assumption does not hold. (2) The lack of strong systematicity in the more "standard" Connectionist architectures points to a greater consideration for behaviour-specific architectural biases. The architectural biases built into the tensor-recurrent network were sufficient to demonstrate strong systematicity on an inference task.

Also discussed was the case where the tensor-recurrent network was too systematic (i.e., generalization across position where it was not evident in the training set). It was suggested that a possible extension to the tensor-recurrent network was in the use of modifiable weights between the tensor and inner product layers, and the use of an additional objective function which encourages the collapsing of the subspaces only in the case where there is significant evidence of overlap in the training set.

Lastly, comments were given on the relationship between the Connectionist tensor-recurrent network solution and the Classical paradigm. The characteristic components of a Classical architecture (i.e., symbols and symbol-level processes) appeared in the tensor-recurrent network as characteristic components of a Connectionist architecture (i.e., vectors and vector operators). Thus, in terms of systematic behaviour, Connectionism offers a "neural-like" implementation of symbolic processing. However, where Connectionism goes beyond Classicism is in an explanation for the acquisition of systematic behaviour, which is an issue not addressed in the Classical paradigm since systematicity is built into a Classical architecture, not acquired.

Chapter 7

Concluding remarks

In **chapter 1**, I introduced two major paradigms for models of cognitive behaviour (i.e., Classicism and Connectionism), and the background to an ongoing debate over the capacity of the Connectionist approach to explain one cognitive-level property called systematicity. Briefly, Fodor and Pylyshyn (1988), and Fodor and McLaughlin (1990) have argued that Connectionist models cannot exhibit properties such as systematicity without implementing a Classical architecture (i.e., without resorting to symbolic representations and processes). Therefore, they have argued that Connectionism is at best an implementation framework for Classical cognitive architectures.

However, as also mentioned in **chapter 1**, this conclusion is still far from unanimously accepted with argument and counter-argument being put forth in an attempt to resolve the issue. Given the lack of consensus, it was decided that a review of the debate over the possibility of a Connectionist (non-classical) explanation of systematicity was an appropriate place to start. Thus, the first question of concern in this thesis was: *Can Connectionist models exhibit systematicity without reliance on symbolic representations and processes?*

7.1 Summary of contribution

The arguments for an alternative Connectionist explanation of systematicity have focused on alternative notions of compositionality (i.e., the way in which architectures construct internal representations which allow them to reason about the external world). The Classical notion of compositionality is that the tokening (i.e., physical inscription) of complex representations is accompanied by a tokening of component representations (i.e., representations of component objects). After reviewing three Connectionist notions of compositionality: Smolensky's weak (microfeatures) and strong (tensors) compositionality; and van Gelder's more general notion of functional compositionality, I concluded, in **chapter 2**, that they fail to provide an alternative explanation of systematicity for one of two reasons, either: (1) component representations are tokened, relative to the processes that access them, wherever complex representations are tokened, in which case one has an implementation of Classical compositionality; or (2) component representations are not tokened, in which case, systematicity of inference cannot be demonstrated since the component representations are not available to other processes. Therefore, I have concluded that Connectionist models cannot provide an alternative explanation for systematicity.

However, I have also argued that the limitation of Fodor and Pylyshyn's thesis is that their requirement for systematicity is grounded solely in terms of computational capacity (i.e., in terms of the functions computable by an architecture). Thus, systematicity can be realized by many computationally sufficient architectures, one of which (i.e., tensors) Fodor and McLaughlin (1990) later reject on the grounds that one must also explain how systematicity is a necessary consequence of the architectural assumptions. However, the limitation of Fodor and McLaughlin's position is that they do not specify a criterion by which a model (Connectionist or otherwise) is said to necessarily exhibit systematicity. Since Connectionism is also concerned with the acquisition of cognitive behaviour, I have suggested that a "potential" contribution of Connectionism to cognitive theory is in an explanation for the necessary acquisition of systematic behaviour. Thus, the second and primary question of concern in this thesis was: *What Connectionist architectural properties are sufficient for the necessary acquisition of systematic behaviour?*

In **chapter 3**, the necessary acquisition of systematic behaviour was defined in terms of probably approximately correct (PAC)-learnability. PAC-learnability is the approximation of a function to within some fixed degree of error with some fixed degree of confidence over repeated trials with resources (e.g., time, processors) polynomial in the size of the function. This criterion was applied to a feedforward network on the auto-association of N-tuples task, designed to test systematicity of representation. The auto-association of N-tuples by a feedforward network has been examined in Brousse and Smolensky (1989) and Brousse (1991). Although Brousse and Smolensky demonstrated an exponential increase in the number of generalizations with respect to N (tuple order), I showed that the percentage of correct to total number of patterns decreased exponentially in N. Therefore, their demonstration of generalization with the feedforward network does not meet the PAC-learnability criterion for the acquisition of systematicity. However, I also showed that with a polynomial (with respect to N) increase in the number of training patterns the feedforward network met the PAC-learnability criterion. The conditions sufficient for the necessary acquisition of systematicity were access to a polynomial number of training examples, and no more than a polynomial number of weights in the network.

PAC-learnability is a useful criterion in that it differentiates possible cognitive architectures on the basis of resource feasibility. Essentially, any architecture requiring more than polynomial resource is impractical as there is rarely enough resource available for even moderately sized behaviours. However, the PAC-learnability criterion did not consider the amount of resource used by people. It is possible for an architecture to acquire systematic behaviour with respect to the PAC-learnability criterion, yet still require more resource than is ever needed by people. Thus, a second criterion based on linguistic evidence, called strong systematicity (Hadley, 1993), was considered in the evaluation of Connectionist models.

Strong systematicity is the ability, above chance level, to correctly represent and process complex objects with components in novel positions (i.e., generalization across position). Based on statistical analysis of training sets, Hadley found no evidence of strong systematicity in the six Connectionist models he examined. It is possible that none of these models exhibited strong systematicity because such a criterion had not been defined at that time, and that the purpose of the Connectionist work in question was just a demonstration of generalization over some structured domain. However, it is also possible that there was some common and fundamental inadequacy with these models given that none of them demonstrated generalization across position.

In **chapter 4**, the feedforward network used in the previous chapter was examined for its capacity to exhibit strong systematicity of representation on the auto-association of 2-tuples task. I showed, by an analysis of the available information and network learning properties, that this feedforward network cannot exhibit strong systematicity of representation. The lack of strong systematicity was attributed to the independence between the weights that implement the mapping of objects in the two component positions. This result suggested that a recurrent network is likely to exhibit strong systematicity of representation, since component objects are presented (extracted) via the same set of input (output) units. Therefore, component mappings are implemented, in part, by the same set of weights. Simulation results showed that a three-layer simple recurrent network can demonstrate strong systematicity of representation on the temporal version of the auto-association of 2-tuples task. Analysis of the internal representations of a trained network suggested that components were mapped to (and from) common internal subspaces. Consequently, generalization across position was achieved as the same set of weights were used to map component representations independent of their position.

The simple recurrent network was then examined for strong systematicity of inference on the querying of 2-tuples task. Simulation results did not show strong systematicity for this network. An analysis of the available information and the learning requirements of the network showed that the network cannot exhibit strong systematicity of inference on this task. The lack of strong systematicity of inference was attributed to the independence between the weights that access components from the two positions of a complex internal representation. It was also argued that strong systematicity of inference cannot be demonstrated in a number of other three-layer networks.

It was suggested that access-weight dependence can be achieved by incorporating an architectural bias that forces or encourages the access of components via a common internal subspace. In **chapter 5**, such a bias was incorporated into a Connectionist architecture called the tensor-recurrent network. In the tensor-recurrent network, tensors were used to represent complex objects. Using the inner product operator, subspaces of the tensor space can be collapsed down to a common internal subspace from which the components can be extracted via a single set of learnable weights. Correct extraction of components, however, requires an appropriate choice of internal component, role and cue vectors, which are assumed to exist in a tensor representational scheme. The novel feature of the tensor-recurrent network is that, by backpropagating an error signal through the units that implement tensor representations, these vectors are learnt by the network. Simulation results showed that this network exhibited strong systematicity of inference on the querying of 3-tuples task.

My design of the tensor-recurrent network was motivated specifically to address the issue of strong systematicity. I have suggested that the network was too strongly biased in that it exhibited some degree of generalization across position when there was no overlap in the training set. In **chapter 6**, I then suggested incorporating an additional error term designed to encourage the collapse of internal subspaces only in the case where there was significant overlap in the training set. In this chapter, I also discussed the implications of the results of this thesis, and the relationship to the Classical paradigm. Specifically, the points made were: (1) The PAC-learning framework, where it relies on the assumption that the training and testing distributions are the same, may be inappropriate for devoloping cognitive modeling in that it can admit architectures (e.g., three-layer feedforward network) that are inadequate when this assumption does not hold (as was the case with respect to strong systematicity). (2) Strong systematicity necessitates more structured (architecturally biased) networks than the three-layer networks examined (e.g., tensor-recurrent networks). (3) The tensor-recurrent network is an example of the contribution of Connectionism as providing an explanation for the necessary acquisition of systematic behaviour from, in part, non-symbolic learning processes.

7.2 Conclusions

The initial question posed in this thesis was:

- ○ *Can Connectionist models exhibit systematicity without reliance on symbolic representations and processes?*

After reviewing the systematicity debate in **chapter 2**, I have concluded that:

- *Connectionist models cannot demonstrate systematicity without implementing some form of symbolic representations and processes.*

From this conclusion it was argued that the problem of systematicity for Connectionism is not in an alternative explanation for systematicity, but in an explanation for the necessary acquisition of systematic behaviour. Thus, the second, and primary question of concern in this thesis was:

- ○ *Can Connectionism provide models that exhibit the necessary acquisition of systematic behaviour?*

After testing and analyzing a number of Connectionist models I have concluded that:

- *Connectionism can provide models, as exemplified by the tensor-recurrent network, with architectural properties that are sufficient for the necessary acquisition of systematic behaviour (i.e., strong systematicity) from, in part, non-symbolic processes.*

Of the Connectionist architectures analyzed, only the tensor-recurrent network demonstrated strong systematicity of inference. From the simulation results and analysis of the tensor-recurrent network in **chapter 5**, I have also concluded that:

- *the architectural properties of the tensor-recurrent network that were, together, sufficient to exhibit strong systematicity on a task designed to test systematicity of inference were:*
  - *tensor units, which construct representations of complex objects as the outer product of internal component and role representations;*
  - *inner product units, which collapse multiple representational subspaces down to a single subspace from which component representations are extracted;*
  - *backpropagation learning dynamics, which allows the appropriate internal component, role and cue representations to be learnt; and*
  - *component-independent, position-dependent generation of role vectors.*

In addressing the two questions of central concern to this thesis, two more general conclusions are drawn. Firstly, in the evaluation of the necessary acquisition of systematicity, two criteria were used: (1) PAC-learnability (**chapter 3**); and (2) Hadley's strong systematicity (**chapters 4 and 5**). The feedforward network demonstrated the acquisition of systematic behaviour as defined by the first criterion, but not the second. Therefore, I conclude that:

- *the PAC-learnability framework, in as far as it assumes that the training and testing distributions are the same, is inappropriate for developing cognitive models that exhibit strong systematicity.*

Secondly, the tensor-recurrent network in **chapter 5** is an instance of a Connectionist architecture where systematic behaviour emerged from initially unsystematic behaviour, partly, as a consequence of a non-symbolic learning process (i.e., backpropagation). Therefore, I conclude that:

- *Connectionism can contribute to cognitive theory in the form of an explanation of the necessary acquisition of symbolic properties such as systematicity from, in part, non-symbolic processes like backpropagation.*

7.3 Further work

The systematicity problem is a watershed for cognitive research in that it segregates potential cognitive architectures on the basis of whether or not such architectures have the capacity to represent and generalize in structured domains.

Systematicity is an instance of a more general problem facing the Connectionist approach to cognitive modeling. That is, learning high-level structured representations from low-level information.

The tensor-recurrent network approach to demonstrating strong systematicity of inference suggests rethinking the notion of the type/token distinction. The advantage of a type/token distinction is that if a function is learnt at the type level then learning generalizes to all tokens of those types. Another way of characterizing the lack of strong systematicity of inference in the first-order networks studied in chapter 4 is that there was no explicit distinction (i.e., one that is readily accessible by other processes within the network) in the network's internal representation space between the tokening of a component and the component's type (i.e., role within the complex object).

The general problem that a type/token distinction introduces into a Connectionist learning framework (which is essentially hill-climbing) is how can a hill-climbing strategy be applied to a domain where identifiers have only logical, not spatial significance. For, in general, the hill-climbing strategy requires a continuously differentiable surface, yet symbol names serve only to identify one symbol from another, and so are, in general, not defined with respect to any similarity measure. That is to say, a comparison between two identifiers is, in general, a two-valued function, either the two identifiers are the same, or they are not the same.

The solution to this problem has been hinted at in the form of the tensor-recurrent network, where type identifiers were vectors in the *role* unit activation space linked (via a continuously differentiable function) to their associated tokens, which were vectors in the *hidden* unit activation space. Since the token and type identifier spaces, and the functions linking the two (i.e., inner and outer products) are all continuously differentiable, type level mappings can be learnt by a hill-climbing strategy through the principle of: *error (information) propagation through structured architectures.*

There are, however, a number of limitations with the tensor-recurrent network, which suggest directions for further work. The first limitation, which was discussed in chapter 6, is the "over-generalization" of the network. The tensor-recurrent network demonstrated some degree of generalization across position when there was no evidence of overlap in the training set. One direction of further work is to incorporate an additional error term so that subspaces are collapsed only in the case where there is significant overlap in the training set. The suggestion given was an error term that reduced the difference between internal representations (vectors) that map to the same output vector. This approach differs from the standard approach that only considers the difference between network and target output.

The second limitation concerns the generation of role vectors. In the tasks considered, the type/token distinction was artificially clean in that each component was associated with one position at each time step. Thus, the point at which to make the binding of a type identifier and its associated token was built into the network, only the values of those identifiers and tokens were learnt. In reality, there are multiple component levels (e.g., sentences are composed of words, and words are composed of phonemes), and the point at which to make/break bindings depends entirely on the structure of the domain. Thus, one possibility for further work is to consider the same tasks, but with input encodings at the phoneme level rather than the word level. A network is still required to demonstrate generalization across word position, but also to learn the points at which to bind component words to their roles.

The inference tasks considered in this thesis only required the networks to extract one of the possible components. More complex behaviours involve the restructuring of input, for example, in active to passive transformations, where complex objects of the form Agent-Action(active)-Patient are mapped to objects with the form Patient-Action(passive)-Agent. One possibility for addressing this task is with modifiable recurrent connections on the *cue* units, which would allow the network to generate a sequence of cue vectors for extraction of a sequence of components. Furthermore, the tasks considered had a flat structure, where each component was an atomic object (i.e., no further components). Natural language processing, however, operates over recursive structures where, for example, a role may be bound to an entire phrase containing its own component-role bindings. Recent work by Hadley (1994), Hadley and Hayward (1994), and Niklasson and van Gelder (1994) consider generalization to different levels of embedding in recursively structured objects as an even tighter criterion for the demonstration of systematicity. In this case, one could include connectivity which allows the binding of role vectors to tensor representations of complex objects, rather than just the hidden unit representations of atomic objects.

More generally, though, people have the capacity to generalize from behaviours learnt in one domain to structurally similar behaviours in other domains. That is, people have the capacity to reason analogically. This degree of generalization goes beyond Hadley's definition of strong systematicity, since in the new domain, people can generalize to complex objects where components have not been seen in any position. For example, given the question, "Grapes are to wine, as rice is to ?", one is likely to respond with "sake", based on knowledge of the relation (i.e., *is-used-in*) in both domains, given very little experience with this type of question before, and almost certainly no experience with the four components in the context of this question. Work has already begun in an attempt to address this form of generalization with Connectionist models (see Halford, Wilson, Guo, Gayler, Wiles, & Stewart, 1994).

In this work, it was assumed that systematicity was not due to any *a priori* similarity between component objects. However, it is possible that existing similarities between component objects (e.g., run and walk) play some part in the acquisition of systematic behaviour. Another possible extension of this work would be to examine networks trained on complex objects whose component representations share some degree of similarity, and then test these networks on objects with completely dissimilar component representations. With respect to the tensor-recurrent network, for example, it may be that initial similarities facilitate the acquisition of stable role vectors for particular component positions.

Connectionist networks have demonstrated generalization on a wide variety of tasks, yet their scalability to the size and complexity of human cognitive domains is still in question. One could argue that the lack of scalability is because of the technology. There is not yet available the processing capacity, in terms of number of processors, in the same order of magnitude as that of the brain. The value of addressing structure-related properties like systematicity is that they also apply to small-scale tasks. Architectures that cannot exhibit systematicity on small tasks will not scale to larger tasks. Thus, research that considers these properties is not only of theoretical significance, but is of practical importance as well.

Appendix A

Understanding as generalization not just representation[1]

Halford and Wilson (1994) define understanding of a concept in terms of representing the relation to which that concept is equivalent. A model is said to have represented a relation when it performs the correct input-output mappings for all functions implicated by that relation. For example, in the balance scale domain, the concept of balance is a relation between two weight and two distance variables and a state variable (which has the possible values: balance, tip-left and tip-right). The concept of balance implicates a number of questions which can be used to evaluate a child's understanding of the concept.

For example, the *balance state* question is a function from the two weights and two distance variables to the state variable. The *missing weight* question is a function from two distance variables, a weight variable, and a state variable to a weight variable. The *missing distance* question is a function from two weight variables, a distance variable and a state variable to a weight variable. More generally, from any combination of four variables, the fifth variable can be predicted.

Halford and Wilson argue that just as a child's understanding of the concept balance is evaluated on a battery of implicated questions, so too a model of that concept can be evaluated by its performance on a number of implicated functions.

[1]This work appears in Phillips (1994).

That is, on the basis of whether or not the model performs the same input-output mappings. Their main point is that such a definition provides an adequate criterion against which candidate Connectionist models can be evaluated. With this definition the authors reject McClelland's (in press) feedforward network model on the basis that it cannot demonstrate adequate performance on all three questions (i.e., the model is incomplete).

The first point I want to make is that implicit in their definition is a notion of generalization (i.e., the capacity to take existing representations and apply them correctly to previously unexperienced situations). I suggest that by confounding these two issues (representation and generalization) some respondents to Halford's presentation have been led to question the authors' justification for rejecting McClelland's feedforward network model.

The authors claim that McClelland's model cannot perform the additional missing weight and missing distance tasks. However, they do not state the basis for this claim. Clearly, McClelland's model cannot be rejected on the basis of the representational capacity of the feedforward network alone. Since multi-layer feedforward networks are universal function approximators (Hornik et al., 1989) there is every reason to expect that given sufficient information and time the network could perform correctly on these additional tasks.

Presumably, however, the authors have some additional criterion in mind. I suspect that although they would agree that McClelland's model (suitably extended) *can* represent the required functions, to do so would require an extensive and unaccountable amount of (re)training. If it can be shown that such training is not available to (nor required by) children, then there is a basis for rejecting the feedforward network model. That is, on the basis of the model's generalization characteristics.

The question is, of course, how much training do children receive? Based on empirical evidence, Hadley (1993, 1994) presents an example of how generalization may be characterized in the linguistic domain. He identifies generalization across syntactic position as a criterion against which Connectionist models can be evaluated. Connectionist models can then be differentiated on the basis of this generalization criterion. For example, as was shown in chapter 4, the three-layer feedforward networks without the assumption of weight tying cannot demonstrate generalization across syntactic position, yet recurrent networks in a limited case can demonstrate this form of generalization.

Essentially, the issue is that although some models have the capacity to represent any function of interest they simply require too many training examples to be psychologically plausible. I suspect that it is on grounds of generalization that the authors want to reject McClelland's model. The point I want to make is that the degree of generalization considered as psychologically plausible must be made explicit; and in doing so, the authors would then have much stronger grounds for rejecting particular Connectionist models. That is, on the basis that these models make use of information either not available to, or not required by children.

The issue of generalization brings me to my second point: if generalization is used as a criterion on which to accept or reject models, then the tensor model that Halford and Wilson are proposing is incomplete. For, although it demonstrates how relational concepts may be represented, there is no story as to how these representations might arise from experience.

In Halford and Wilson's formulation, an appropriate arrangement of weights and connections are in place to represent relational concepts as tensor products. Presumably, however, these weights and connections were not always there otherwise this model could not account for the empirical fact that children below a certain developmental stage do not understand (in Halford and Wilson's sense) the concept of balance.

The two issues of representation and generalization place the authors in a dilemma. Explicitly, they want to distinguish between two models on the basis of representational capacity, yet both models (suitably extended) are capable of representing relations. Implicitly, I suspect they want to reject McClelland's model on grounds of generalization. However, if generalization is the distinguishing criterion then they are required to tell a story of how the *right* arrangement of weights and connections of a tensor model come into being; and that is a story not yet told.

Bibliography

Abu-Mostafa, Y. S. (1990). Learning from hints in neural networks. *Journal of Complexity*, *6*, 192–198.

Al-Mashouq, K. A., & Reed, I. S. (1991). Including hints in training neural nets. *Neural Computation*, *3*, 418–427.

Amsterdam, J. (1988). Some philosopical problems with formal learning theory. In *The seventh national conference on artifical intelligence*, pp. 580–584. Saint Paul, MN: Morgan Kaufmann.

Anthony, M. (1992). *An introduction to computational learning theory.* Cambridge tracts in theoretical computer science. Cambridge, England: Cambridge University Press.

Bakker, P., Phillips, S., & Wiles, J. (1993). The N-2-N encoder: A matter of representation. In S. Gielen & B. Kappen (Eds.), *ICANN'93: Proceedings of the International Conference on Artificial Neural Networks*, pp. 554–557. London: Springer-Verlag.

Bakker, P., Phillips, S., & Wiles, J. (1994). The 1000-2-1000 encoder: A matter of representation. *Neural Network World*, *4*(5), 527–534.

Bartlett, P. L. (1992). *Computational learning theory and neural network learning.* Ph.D. thesis, The University of Queensland, Dept. of Electrical and Computer Engineering.

Baum, E. B., & Haussler, D. (1989). What size net gives valid generalization? *Neural Computation*, *1*, 151–160.

Baum, E. B., & Wilczek, F. (1987). Supervised learning of probability distributions by neural networks. In D. Z. Anderson (Ed.), *Neural Information Processing Systems*, pp. 52–61. New York, NY: American Institute of Physics.

Blum, A. L., & Rivest, R. L. (1990). Training a 3-node neural network is NP-complete. Tech. Rep., MIT, Cambridge, MA.

Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, *5*(1), 117–127.

Brooks, R. A. (1991a). Intelligence without reason. In *12th International Joint Conference on Artificial Intelligence*, pp. 569–595.

Brooks, R. A. (1991b). Intelligence without representation. *Artificial Intelligence*, *47*, 139–160.

Brousse, O. J. (1991). *Generativity and systematicity in neural network combinatorial learning*. Ph.D. thesis, University of Colorado, Boulder, CO.

Brousse, O. J., & Smolensky, P. (1989). Virtual memories and massive generalization in connectionist combinatorial learning. In *Proceedings of the 11th Annual Conference of the Cognitive Science Society*, pp. 380–387. Hillsdale, NJ: Lawrence Erlbaum.

Brown, T. L., & LeMay, H. E. (1981). *Chemistry: The central science* (2nd edition). Englewood Cliffs, NJ: Prentice Hall.

Butler, K. (1993). Connectionism, classical cognitivism and the relation between cognitive and implementational levels of analysis. *Philosophical Psychology*, *6*(3), 321–333.

Chalmers, D. J. (1990a). Syntactic transformations on distributed representations. *Connection Science*, *2*, 53–62.

Chalmers, D. J. (1990b). Why Fodor and Pylyshyn were wrong: The simpliest refutation. In *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society*, pp. 340–347. Hillsdale, NJ: Lawrence Erlbaum.

Chalmers, D. J. (1993). Connectionism and compositionality: Why Fodor and Pylyshyn were wrong. *Philosophical Psychology*, *6*, 305–319.

Charter, N., & Oaksford, M. (1990). Autonomy, implementation and cognitive architecture: A reply to Fodor and Pylyshyn. *Cognition*, *34*, 93–107.

Clark, A. (1991). Systematicity, structured representations and cognitive architecture: A reply to Fodor and Pylyshyn. In T. Horgan & J. Tienson (Eds.), *Connectionism and the Philosophy of Mind*, pp. 198–218. Dordrecht, The Netherlands: Kluwer Academic.

Dennis, S., & Phillips, S. (1991). Analysis tools for neural networks. Tech. Rep. 207, University of Queensland, Brisbane, Australia.

Dyer, M. G. (1991). Connectionism versus symbolism in higher-level cognition. In T. Horgan (Ed.), *Connectionism and the Philosphy of Mind*, pp. 382–416. Dordrecht, The Netherlands: Kluwer Academic.

Elman, J. L. (1989). Representation and structure in connectionist models. Tech. Rep. 8903, University of California, San Diego, CA.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, *14*, 179–211.

Elman, J. L. (1991). Incremental learning, or the importance of starting small. Tech. Rep. 9101, University of California, San Diego, CA.

Fodor, J. A. (1975). *The Language of Thought*. Language and Thought. New York, NY: Crowell.

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

Fodor, J. A., & Pylyshyn, Z. W. (1988). Connectionism and cognitive architecture: A critical analysis. *Cognition*, *28*, 3–71.

Funahashi, K. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, *2*, 183–192.

Garey, M. R., & Johnson, D. S. (1979). *Computers and Intractability: A guide to the theory of NP-completeness.* San Francisco, CA: W H Freeman.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation*, *4*(1), 1–58.

Hadley, R. F. (1993). Compositionality and systematicity in connectionist language learning. Tech. Rep. CSS-IS TR 93-01, Simon Fraser University, Burnaby, BC.

Hadley, R. F. (1994). Systematicity in connectionist language learning. *Mind and Language*, *9*(3), 247–272.

Hadley, R. F., & Hayward, M. (1994). Strong semantic systematicity from unsupervised connectionist learning. Tech. Rep. CSS-IS TR94-02, Simon Fraser University.

Halford, G. S., & Wilson, W. H. (1994). How far do neural networks account for human reasoning? In J. Wiles, C. Latimer, & C. Stevens (Eds.), *Collected Papers from a symposium on connectionist models and psychology*, pp. 50–62. Department of Computer Science, The University of Queensland, Brisbane, Australia.

Halford, G. S., Wilson, W. H., Guo, J., Gayler, R. W., Wiles, J., & Stewart, J. E. M. (1994). Connectionist implications for processing capacity limitations in analogies. In K. J. Holyoak & J. Barnden (Eds.), *Advances in Connectionist and Neural Network theory*, chap. 7, pp. 363–415. Norwood, NJ: Ablex.

Hartman, E., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with gaussian hidden units as universal approximators. *Neural Computation*, *2*(2), 210–215.

Haussler, D. (1992). Decision theoretic generalizations of the PAC model for neural net and other learning applications. Tech. Rep. UCSC-CRL-91-02, University of California, Santa Cruz, CA.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Redwood City, CA: Addison-Wesley.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*, 359–366.

Jordan, M. I. (1990). Serial order: A parallel distributed processing approach. Tech. Rep., MIT, Cambridge, MA.

Judd, S. (1987). Learning in networks is hard. In M. Caudill & C. Butler (Eds.), *IEEE International Conference on Neural Networks*, pp. 685–692. Piscataway, NJ: IEEE Service Center.

Judd, S. (1988). On the complexity of loading shallow neural networks. *Journal of Complexity*, *4*, 177–192.

Judd, S. (1990). *Neural Network Design and the Complexity of Learning*. Cambridge, MA: MIT Press.

Karmiloff-Smith, A. (1992). *Beyond Modularity: A developmental perspective on cognitive science*. Cambridge, MA: MIT Press/Bradford Books.

Kern, L. H., Mirels, H. L., & Hinshaw, V. G. (1983). Scientists' understanding of propositional logic: An experimental investigation. *Social Studies of Science*, *13*, 131–146.

Kirsh, D. (1991a). Today the earwig, tomorrow man. *Artificial Intelligence*, *47*(1), 161–184.

Kirsh, D. (1991b). When is information explicitly represented? In P. Hanson (Ed.), *Information, Language and Cognition: Vancouver Studies in Cognitive Science*, pp. 340–365. Vancouver, BC: UBC Press.

Kruglyak, L. (1990). How to solve the N bit encoder problem with just two hidden units. *Neural Computation*, *2*, 399–401.

Lapedes, A., & Farber, R. (1988). How neural nets work. In D. Z. Anderson (Ed.), *Neural Information Processing Systems*, pp. 442–456. New York, NY: American Institute of Physics.

LeCun, Y., Boser, B., & Denker, J. S. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

Lin, J. H., & Vitter, J. S. (1989). Complexity issues in learning by neural nets. In R. Rivest, D. Haussler, & M. K. Warmath (Eds.), *Proceedings of the Second Annual Workshop on Computational Learning Theory*, pp. 118–133. San Mateo, CA: Morgan Kaufmann.

Lister, R. (1993). Visualizing weight dynamics in the N-2-N encoder. In *Proceedings of the IEEE International Conference on Neural Networks*. Piscataway, NJ: IEEE Service Center.

Lister, R., Bakker, P., & Wiles, J. (1993). Error signal, exceptions and back propagation. In *Proceedings of the 1993 International Joint Conference on Neural Networks*, pp. 573–576. Piscataway, NJ: IEEE Service Center.

Marcus, G. F., Brinkmann, U., Clahsen, H., Wiese, R., Woest, A., & Pinker, S. (1993). German inflection: The exception that proves the rule. Tech. Rep. 47, MIT, Cambridge, MA.

Marr, D. (1982). *Vision: a computational investigation into the human representation and processing of visual information.* San Francisco, CA: W H Freeman.

Maskara, A., & Noetzel, A. (1992). Forcing simple recurrent networks to encode context. Tech. Rep., New Jersey Institute of Technology, NJ. Also in the Proceedings of the 1992 Long Island Conference on Artificial Intelligence and Computer Graphics.

McClelland, J. L. (in press). A connectionist perspective on knowledge and development. In T. Simon & G. S. Halford (Eds.), *Developing Cognitive Competence: New Approaches to Cognitive Modeling*. Hillsdal, NJ: Erlbaum.

McClelland, J. L., & Kawamoto, A. H. (1986). *Mechanisms of Sentence Processing: Assigning roles to Constituents of Sentences*, Vol. 2 of *Computational models of cognition and perception*, chap. 19, pp. 272–331. Cambridge, MA: MIT Press.

McClelland, J. L., Rumelhart, D. E., & the PDP research group (Eds.). (1986). *Parallel Distributed processing: Explorations in the microstructure of cogntion*, Vol. 2 of *Computational models of cognition and perception*. Cambridge, MA: MIT Press.

Minsky, M. L., & Papert, S. A. (1990). *Perceptrons* (4th edition). Cambridge, MA: MIT Press.

Newell, A. (1980). Physical symbol systems. *Cognitive Science*, *4*, 135–183.

Niklasson, L., & Sharkey, N. E. (1992). Connectionism and the issues of compositionality and systematicity. In R. Trappl (Ed.), *Proceedings of the Cybernetics and Systems Research*, pp. 1367–1374.

Niklasson, L., & van Gelder, T. (1994). Can connectionist models exhibit nonclassical structure sensitivity. In A. Ram & K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 644–669. Lawrence Erlbaum.

Niklasson, L. F. (1993). Structure sensitivity in connectionist models. In *Proceedings of the 1993 Connectionist Summer School*, pp. 162–169. Hillsdale, NJ: Lawrence Erlbaum.

Nowlan, S. J., & Hinton, G. E. (1991). Simplifying neural networks by soft-weight sharing. Tech. Rep., The Salk Institute, San Diego, CA.

Phatak, D. S. (1993). Construction of minimal n-2-n encoders for any n. *Neural computation*, *5*(5), 783–794.

Phillips, S. (1993). The effect of representation on error surface. In P. Leong & M. Jabri (Eds.), *Proceedings of the Fourth Australian Conference on Neural Networks*, pp. 86–89. Sydney, Australia: Sydney University Electrical Engineering.

Phillips, S. (1994). Understanding as generalization not just representation. In J. Wiles, C. Latimer, & C. Stevens (Eds.), *Collected Papers from a symposium on connectionist models and psychology*, pp. 110–111. The University of Queensland, Brisbane, Australia: Department of Computer Science. Technical report No. 289. A comment on Halford and Wilson's 'How far do neural networks account for human reasoning?'.

Phillips, S., & Wiles, J. (1991). On learning long-term contingencies. Unpublished manuscript.

Pinker, S. (1984). *Language Learnability and Language Development.* Cognitive Science Series. Cambridge, MA: Harvard University Press.

Pinker, S., & Prince, A. (1988). On language and connectionism: Analysis of a parallel distributed processing model of language acquisition. *Cognition*, *28*, 73–193.

Pollack, J. B. (1990). Recursive distributed representations. *Artificial Intelligence*, *46*, 77–105.

Pylyshyn, Z. W. (1980). Computation and cognition: Issues in the foundations of cognitive science. *Behavioral and Brain Sciences*, *3*, 111–169.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning Internal Representations by Error Propagation*, Vol. 1 of *Computational models of cognition and perception*, chap. 8, pp. 319–362. Cambridge, MA: MIT Press.

Rumelhart, D. E., & McClelland, J. L. (1986). *On Learning Past Tenses of English Verbs*, chap. 18, pp. 216–271. Computational models of cognition and perception. Cambridge, MA: MIT Press.

Rumelhart, D. E., McClelland, J. L., & the PDP research group (Eds.). (1986). *Parallel Distributed processing: Explorations in the microstructure of cogntion*, Vol. 1 of *Computational models of cognition and perception.* Cambridge, MA: MIT Press.

Sato, M., Murakami, Y., & Joe, K. (1990). Learning chaotic dynamics by recurrent neural networks. In *Proceedings of the International Conference on Fuzzy Logic and Neural Networks.*

Shawe-Taylor, J., & Anthony, M. (1991). Sample sizes for multiple-output threshold networks. *Network, Computation in Neural Systems*, *2*(1), 107–117.

Siegelman, H. T., & Sontag, E. D. (1991). On the computational power of neural nets. Tech. Rep., Rutgers University, New Brunswick, NJ.

Smolensky, P. (1987a). The constituent structure of connectionist mental states: A reply to Fodor and Pylyshyn. *Southern Journal of Philosophy*, *26*, 137–161.

Smolensky, P. (1987b). On variable binding and the representation of symbolic structures in connectionist systems. Tech. Rep. CU-CS-355-87, University of Colorado, Boulder, CO.

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

Smolensky, P. (1991). Connectionism, constituency, and the language of thought. In B. Loewer & G. Rey (Eds.), *Meaning in Mind: Fodor and his critics*, chap. 12, pp. 201–227. Cambridge, MA: Blackwells.

Spivey, J. M. (1989). *The Z Notation: A reference manual.* Computer Science. New York, NY: Prentice Hall.

St. John, M. F., & McClelland, J. L. (1990). Learning and applying contextual constraints in sentence comprehension. *Artificial Intelligence*, *46*, 217–257.

Suddarth, S. C., & Holden, A. D. C. (1991). Symbolic-neural systems and the use of hints in developing complex systems. *International Journal of Man-Machine Studies*, *35*, 291–311.

Suddarth, S. C., & Kergos, Y. L. (1990). Rule injection hints as a means of improving network performance and learning time. In L. B. Almeida & C. J. Wellekens (Eds.), *Neural Networks*, Vol. 412 of *Lectures in Computer Science*, pp. 120–129. New York, NY: Springer-Verlag.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, *27*(11), 1134–1142.

van Gelder, T. (1990). Compositionality: A connectionist variation on a classical theme. *Cognitive Science*, *14*, 355–384.

van Gelder, T., & Niklasson, L. (1994). Classicism and cognitive architecture. In A. Ram & K. Eiselt (Eds.), *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 905–909. Lawrence Erlbaum.

Wiles, J., & Bloesch, A. (1992). Operators and curried functions: Training and analysis of simple recurrent networks. In S. J. Hanson & R. P. Lippman (Eds.), *Advances in Neural Information Processing Systems 4*. San Mateo, CA: Morgan Kaufmann.

Wiles, J., & Ollila, M. (1993). Intersecting regions: the key to combinatorial structure in hidden unit space. In S. J. Hanson, J. D. Cowan, & C. L. Giles (Eds.), *Advances in Neural Information Processing Systems 5*, pp. 27–33. San Mateo, CA: Morgan Kaufmann.

Wiles, J., Phillips, S., & Norris, M. (1993). Generalization and training complexity in a combinatorial domain. Unpublished manuscript.

Zemel, R. S. (1993). *A minimum description length framework for unsupervised learning.* Ph.D. thesis, University of Toronto, Toronto, Canada.