The University of Sussex

Dynamics of Arithmetic
A Connectionist View of Arithmetic Skills

Richard Dallaway

Submitted for the degree of D. Phil.

August 1993

UNIVERSITY OF

SUSSEX

AT BRIGHTON

Contents

Acknowledgements

Many thanks to my supervisors, Harry Barrow & David Young—not just for their role in shaping this thesis, but for the education in general.

I thank my friends who provided a good mix of help and distraction, especially: Julian Budd, Teresa del Soldato, Inman Harvey, Jo Brook, Paulo Costa, Ben du Boulay, Geoff Goodhill, Suran Goonatilake, Richard Hall, Sharon Groves, and Paul Booth.

Julia kept me sane while living through the best and worst of it herself: from near-death in Pittsburgh to nights by the West Pier. Thanks Jules.

A big thank-you to my family for all sorts of support on all sorts of occasions.

I specifically acknowledge: Mike McCloskey & Grethe Lindemann for their comments and hospitality; Alistair Chalmers for helping out with some of the stats; Angela Jackson of the Education Development Building for explaining how schools teach arithmetic; David Bennett for explaining the BSB model; Judith Henstra at Experimental Psychology for explaining a few things about eye movements; and Shaaron Ainsworth for being, as she mistakenly called herself, "the enemy".

This work was funded by a SERC award in conjunction with Integral Solutions Ltd. Thanks to Julian Clinton and Alan Montgomery for their support. The simulations reported in this thesis were performed with a modified version of the McClelland & Rumelhart (1988) *bp* program, and the analysis was carried out in POPLOG POP-11.

Declaration

I hereby declare that this thesis has not been submitted, either in the same or different form, to this or any other university for a degree.

Richard Dallaway

The University of Sussex

Dynamics of Arithmetic: A Connectionist View of Arithmetic Skills

Richard Dallaway

Submitted for the degree of D. Phil., August 1993

Abstract

Arithmetic takes time. Children need five or six years to master the one hundred multiplication facts (0×0 to 9×9), and it takes adults approximately one second to recall an answer to a problem like 7×8. Multicolumn arithmetic (e.g., 45×67) requires a sequence of actions, and children produce a host of systematic mistakes when solving such problems. This thesis models the time course and mistakes of adults and children solving arithmetic problems. Two models are presented, both of which are built from connectionist components.

First, a model of memory for multiplication facts is described. A system is built to capture the response time and slips of adults recalling two digit multiplication facts. The phenomenon is thought of as spreading activation between problem nodes (such as "7" and "8") and product nodes ("56"). The model is a multilayer perceptron trained with backpropagation, and McClelland's (1988) cascade equations are used to simulate the spread of activation. The resulting reaction times and errors are comparable to those reported for adults. An analysis of the system, together with variations in the experiments, suggest that problem frequency and the "coarseness" of the input encoding have a strong effect on the phenomena. Preliminary results from damaging the network are compared to the arithmetic abilities of brain-damaged subjects.

The second model is of children's errors in multicolumn multiplication. Here the aim is not to produce a detailed fit to the empirical observations of errors, but to demonstrate how a connectionist system can model the behaviour, and what advantages this brings. Previous production system models are based on an *impasse-repair* process: when an child encounters a problem an impasse is said to have occurred, which is then repaired with general-purpose heuristics. The style of the connectionist model moves away from this. A simple recurrent network is trained with backpropagation through time to activate procedures which manipulate a multiplication problem. Training progresses through a curriculum of problems, and the system is tested on unseen problems. Errors can occur during testing, and these are compared to children's errors. The system is analysed in terms of hidden unit activation trajectories, and the errors are characterized as "capture errors". That is, during processing the system may be attracted into a region of state space that produces an incorrect response but corresponds to a similar arithmetic subprocedure. The result is a *graded state machine*—a system with some of the properties of finite state machines, but with the additional flexibility of connectionist networks. The analysis shows that connectionist representations can be structured in ways that are useful for modelling procedural skills such as arithmetic. It is suggested that one of the strengths of the model is its emphasis on development, rather than on "snap-shot" accounts. Notions such as "impasse" and "repair" are discussed from a connectionist perspective.

CHAPTER 1

Introduction

Simple arithmetic skills are difficult to master. Although we understand arithmetic well intellectually, we falter in its execution. As Marr put it: "I have no doubt that when we do mental arithmetic we are doing something well, but it is not arithmetic" (1982, p. 348). Children have acquired a host of impressive skills by the time they are taught formal arithmetic: they have learned a language and can use vision for autonomous navigation in a hostile environment. In contrast, the "simple" tasks of arithmetic require at least a further five or six years of schooling. Once the skills are learned there are many opportunities for error. Adults, for example, make plenty of mistakes recalling multiplication facts—especially on the "tricky" problems, such as 8×4 or 9×8. Arithmetic, it seems, is not an easy skill to come by.

So what is the "something" that we do well when we solve arithmetic problems? The view taken here is that the things we do well are those tasks suited to a certain kind of computation—namely connectionism. Difficult tasks, such as arithmetic, need to be turned into pattern matching problems. That is, "we succeed in solving logical problems not so much through the use of logic, but by making the problems we wish to solve conform to problems we are good at solving" (Rumelhart, Smolensky, McClelland & Hinton 1986, p. 44). Exactly how this is done for arithmetic is the topic of this thesis. Two elementary arithmetic skills are considered: adult memory for multiplication facts and children's errors in long (multicolumn) multiplication.

1.1 Part I—Mental arithmetic

The first part of the thesis considers recall of multiplication facts, such as 3×6=18. Intuitively it seems that recalling the answer to such a problem is neither difficult nor drawn-out. However, the model presented here suggests a brief struggle between products in memory before the (often correct) answer "pops up." As problems get more difficult, which usually means larger, the struggle takes a little longer—around one second in some cases. Certain problems are easier than others, notably the tie problems (2×2, 3×3, and so on), and the 5s problems (5 times anything). When mistakes are made they tend to be related to the presented problem. For example, a common mistake is 6×7=48, and 48 is the answer to 6×8. That is, errors are often the correct answer for a problem which shares an operand with the presented problem.

This phenomenon and previous models are looked at in detail in chapter 2. The review draws on normal and brain-damaged studies of the reaction times and errors of adults recalling multiplication facts.

Chapter 3 describes the connectionist model built to capture the reaction times and errors of adults. The basic idea is a simple one: memory for multiplication facts consists of a set of associations between operands and products; recall is the process of spreading activation, resulting in a product's activation exceeding a threshold. The activation spreads at different rates for different problems, giving different reaction times. Occasionally, when under some kind of time pressure, a false product exceeds the threshold. Most of these errors are operand errors, and the reasons for this are explored.

By varying the assumptions of the simulations, certain factors were found to be important in determining the phenomena. There is some evidence that smaller problems are experienced more often than larger problems, and this skew in frequency has a strong affect on the model. Also, the input encoding to the network (representing the operands of a problem) can effect the distribution of errors and reaction times. In particular, the degree of "coarseness" or "sharpness" of the encoding is explored.

Other phenomena are also investigated. For example, it seems that zero problems (zero times anything) are solved by the rule 0×N=0. There is plenty of evidence for this, including: zero problems are solved very quickly; errors are of the form 0×N=N;

brain-damaged patients can re-learn zero problems from exposure to just 2 examples, but not non-zero problems. There is certainly something special about zero, and it is not clear how this fits into the associative framework.

1.2 *Part II—Multicolumn multiplication*

The second half of the thesis examines children's errors on multicolumn multiplication problems. Behaviour on these problems seems to be rule governed. Children pick up a collection of "bugs"—systematic perturbations to the correct rules of arithmetic— and apply the rules producing all sorts of errors. For example:

$$\begin{array}{cccc} & 5 & 2 & 4 \\ \times & 7 & 3 & 1 \\ \hline 3 & 5 & 6 & 4 \end{array} \qquad\qquad \begin{array}{ccc} & 7 & 6 \\ \times & & 4 \\ \hline 1 & 4_2 & 4 \end{array}$$

In the first example, the child multiplies using the pattern for addition: $1\times4=4$, $3\times2=6$, $7\times5=35$. The second example shows a child getting the first multiplication, $4\times6=24$, correct. Then, when there is no second multiplier, the child uses the carry in the next multiplication: $2\times7=14$.

These errors have previously been modelled with production systems. Chapters 4 reviews the literature on buggy behaviour and models of buggy behaviour, and outlines relevant work from connectionism. Particular attention is paid to VanLehn's (1990) "Sierra" model, as this seems to be the best available model of procedural misconceptions (see also Pirolli 1991). Briefly, the prevailing notion is that children reach "impasses" when solving problems—situations in which no rules directly apply. These impasses need to be "repaired" by general purpose heuristics. Sierra has these heuristics and a learning mechanism. Incomplete arithmetic rules are learned, which means that Sierra reaches impasses. Different errors are observed depending on what kind of repair is carried out.

Sierra is a successful model, and the mistakes children make they do seem to derive from following faulty rules. How could a connectionist build a model of this behaviour? As Boden (1988, p. 167) notes:

> It is not clear that processes of relaxation using multiple constraints, powerful though they may be for pattern matching, are well suited to modelling con-

scious planning or cryptarithmetic—or even mere arithmetic, for that matter.

Chapter 5 explains the approach taken in giving a connectionist interpretation of multicolumn arithmetic.

A set of operations was devised to allow a network to move around, read from and write on a problem. A "curriculum" of problems was selected, starting with easy addition tasks and moving up to three column multiplication. Each problem was encoded as a sequence of operations and a recurrent network was trained to activate the correct operation at the correct moment when solving a problem. Buggy behaviour was exhibited when the network was tested on unseen problems. Analysis shows that the representations learned by the model have a procedural structure, allowing bugs to be composed of correct skills plus chunks of skills which are correct in other situations. It is suggested that the gradual learning of these representations is an interesting alternative to snap-shot rule acquisition accounts.

Connectionist models do not reach impasses as such. That is, there is never a moment when the system "gets stuck" and needs to repair the current state. Hence it is necessary to address the role of impasses in learning this task. Are impasses important learning events, or just a by-product of the problem solving mechanism?

It was never going to be possible to build a model which could compete with the empirical power of VanLehn's system: Sierra is the product of over ten years research. Rather, the work described in this half of the thesis is best thought of as a "demonstrator" of how one might model arithmetic from a connectionist perspective.

1.3 Structure of arithmetic skills

The splitting of arithmetic skills into two models—fact recall and procedural skills—is supported by studies of brain-damaged subjects (McCloskey & Caramazza 1985; McCloskey, Aliminosa & Sokol 1991). Figure 1.1 shows the structure of the number-processing system. This structure was devised by noting that particular components of the system can be selectively damaged. For example, one subject (RR) was asked to read Arabic numbers aloud. For 37 000 he said "Fifty-five thousand", for 2 he said "one" (McCloskey & Caramazza 1985, pp. 187–188). Yet RR could determine which of two presented

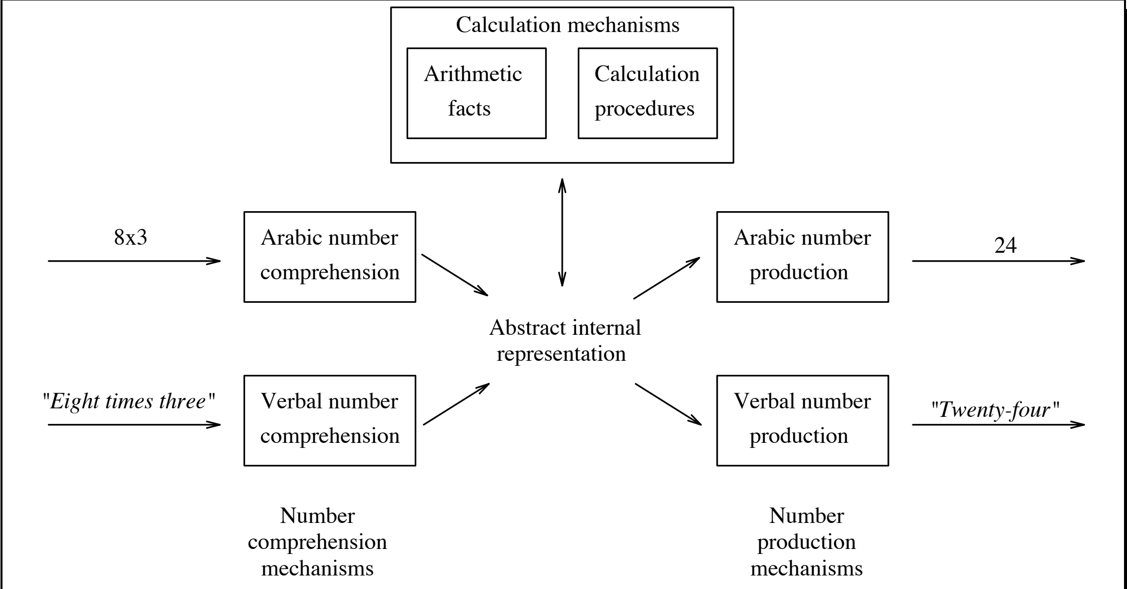
Calculation mechanisms
Arithmetic facts
Calculation procedures
8x3
Arabic number comprehension
Arabic number production
24
Abstract internal representation
"Eight times three"
Verbal number comprehension
Verbal number production
"Twenty-four"
Number comprehension mechanisms
Number production mechanisms

Figure 1.1. The structure of the cognitive number processing system (after McCloskey, Aliminosa & Sokol 1991, figure 1).

numbers was larger, and had no trouble selecting a pile of tokens that corresponded to a presented Arabic number. It seems that RR had no difficulty in comprehending and representing number, but was impaired in production alone.

This, and other experiments, led McCloskey & Caramazza (1985) to propose the structure shown in figure 1.1. The model of arithmetic memory deals with the "arithmetic facts" and "abstract representation" parts of the figure. The multicolumn model is concerned with the "calculation procedures" part of the structure.

1.4 Aims

There are two major aims. First, to build an explicitly specified model of memory for multiplication facts. Previous models have been poorly specified—either not implemented at all, or making assumptions such as the probability of an error being proportional to answer node activation. These details need to be fleshed out in order to understand the importance of various assumptions, or changes to assumptions. Hence, the first contribution is an explicit model that can be tested and criticised. Variations on the model aim to understand the causes of the phenomenon. The causes include the frequency of problems, the creation of false associations and the nature of the facts themselves.

The second aim is to demonstrate an alternative to production system models of multicolumn arithmetic, and to show that such an approach is useful. This constitutes the first connectionist model of this phenomena. Errors are characterized as perturbations to processing trajectories, rather than faulty rules or repairs to impasses. This view conceptualizes learning as the formation and differentiation of states in something similar to a finite state machine. In addition, the analysis of the system is a useful analysis of a sequential network learning a large structured problem.

Part I

Mental Arithmetic

CHAPTER 2

Memory for Arithmetic Facts

There are a number of ways to find the answer to “6×7”. Strategies might include counting a row of six 7s, recalling the answer to 6×6 and adding on another 6, using a calculator, or pure recall. Children tend to use a number of strategies, but as they become older they tend to rely on recall alone (Siegler 1988).

This chapter investigates adults’ recall of multiplication facts. Although adults do use other strategies, recall seems to be most frequently used, and it is also the strategy that has been the subject of many detailed experiments. First, a review is presented of the typical reaction times (RTs) and errors of adults recalling multiplication facts. A number of models have been proposed to account for the phenomena, and these are reviewed in section 2.2. A new connectionist model of fact recall, based on McClelland’s (1979, 1988) “cascade” equations, is described in chapter 3.

2.1 Phenomena

When asked to recall answers to single-digit multiplication problems, both children and adults exhibit well documented patterns of behaviour. These behaviours are recorded from experiments based around three kinds of task: production, verification, and primed production. In the production task, subjects are presented with two digits and asked to recall the product. The primed production task is similar to the production task, but before the two digits are presented, the subject is shown a number which may or may not be the correct answer to the problem. For the verification task the subject is presented with a problem and candidate solution ("6×7=48?") and has to decide if the equation is true or false. In all cases, errors and RTs are recorded.

As production is the every-day task that subjects are familiar with, it is the one that is considered here. Many of the experimental results come from normal subjects (e.g., Campbell & Graham 1985; Miller, Permutter & Keating 1984; Campbell 1987; Ashcraft 1982; Siegler 1988; Harley 1991; Krueger 1986). However, there are interesting results from brain-damaged subjects (McCloskey, Aliminosa & Sokol 1991; Sokol, McCloskey, Cohen & Aliminosa 1991; McCloskey & Caramazza 1985), and these are considered in section 2.1.2.

2.1.1 *The production task*

Typical production experiments (e.g., Campbell & Graham 1985) run as follows: a subject is seated in front of a computer screen on which two digits will appear. The subject is asked to respond "as quickly and accurately as possible" when the digits are shown. The RT is recorded along with any errors the subject makes. Usually only the problems from 2×2 to 9×9 are tested. In some experiments 0×0 to 9×9 is tested, although zero and ones problems are often assumed to be solved by a separate mechanism. The complications added by zero and ones problems are discussed later.

Reaction time

In general, RTs increase across the multiplication tables. That is, problems in the nine times table tend to take longer to answer than problems in the two times table. However, this "problem-size effect" has exceptions:

- The five times table is much faster than its position would suggest.
- "Tie" problems (2×2, 3×3 etc.) are recalled relatively quickly.

Figure 2.1 shows the RTs of adults and grade 5 children for the eight multiplication tables, 2–9. Overall, the developmental trend is for a flattening of the RT curve, and a increase in response speed. Note that for the grade 5 subjects there is a noticeable dip in RT for the nine times table which is not present for adults. Zero and ones problems are considered later (section 2.1.3).

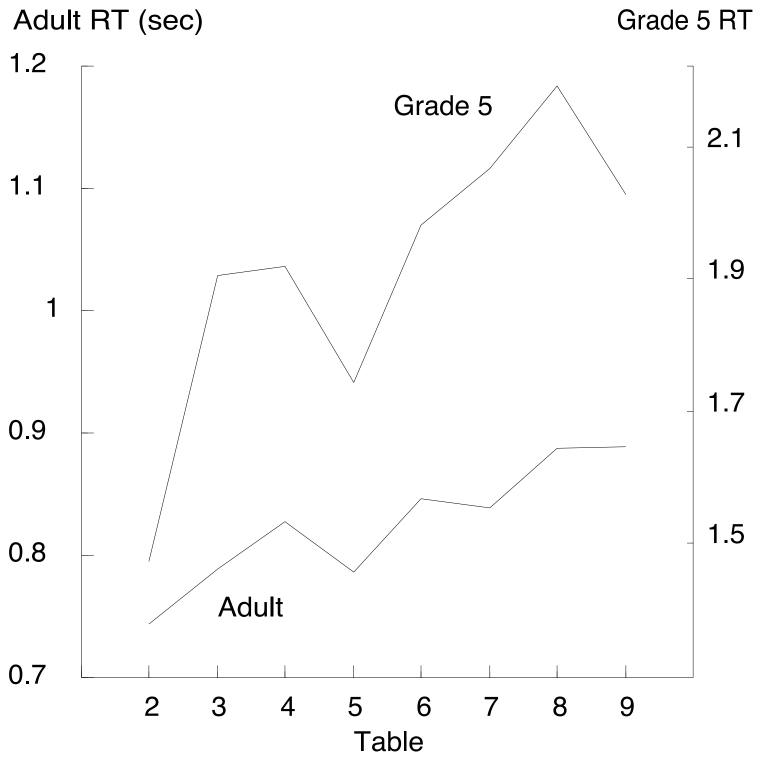

Figure 2.1. Plot of mean correct RT per multiplication table collapsed over operand order for mean RT of: 60 adults (Campbell & Graham 1985, appendix A); 26 children in grade 5 (ibid., appendix B).

Errors

Campbell & Graham (1985) found that adults under mild time pressure make errors at the rate of 7.65 per cent. These errors are distributed as shown in table 2.1. Errors tend to be clustered around the correct product. More specifically, the errors can be classified as follows (after McCloskey, Harley & Sokol 1991):

- Operand errors, for which the erroneous product is correct for a problem that shares a digit (operand) with the presented problem. For example, 8×8=40 is an operand error because the problem shares an operand, 8, with 5×8=40.
- Close operand errors, a subclass of operand errors, where the erroneous product is also close in magnitude to the correct product. That is, for the problem $a \times b$, the error will often be correct for the problem $(a \pm 2) \times b$ or $a \times (b \pm 2)$. An example is 5×4=24. This phenomenon is referred to as the "operand distance effect".

	4	6	8	9	10	12	14	15	16	18	20	21	24	25	27	28	30	32	35	36	40	42	45	48	49	54	56	63	64	72	81
2×2	c																														
2×3		c																													
3×2		c				1																									
2×4	1	1	c		1	1	1																								
4×2			c			2			1																						
2×5					c	1		2			1																				
5×2					c	2																									
2×6		3				c				2																					
6×2		1				c			1																						
2×7						1	c			1		2			1																
7×2				1		2	c		1	1																					
2×8						1	1		c	8																					
8×2						2	1		c	3																					
2×9										c																					
9×2									2	c																					
3×3		5		c																											
3×4	1					c																									
4×3						c																									
3×5								c						1			1														
5×3								c																							
3×6				1		2			1	c										2											
6×3						2				c		2	2		1			1													
3×7												c			4																
7×3							1					c			1																
3×8									2	10		1	c		1																
8×3									1	5			c		3			1													
3×9				2						8		7	1		c					1											
9×3										13		4	1		c					2											
4×4	1	3				2			c	2			1																		
4×5	1			1	1						c										1										
5×4				1	1						c																				
4×6													c			1				2											
6×4													c			1															
4×7												3	3		1	c		1			1				2						
7×4													6			c				2		1		1							
4×8													15			3		c		5				4							
8×4													20			2		c				2							2		
4×9													1		2			4		c	1				2						
9×4											1		1		3	5		1		c											
5×5														c					3												
5×6														1			c		3	1							1				
6×5																	c	1	5		1										
5×7																	1	1	c			1	4			1	1				
7×5																	2		c				2								
5×8											1			1					2		c		3	1			1				
8×5																	2		2		c		4								
5×9																2			3		6		c		1	5					
9×5																			2		4		c			1					
6×6													1							c	1	1						1	1		
6×7																1		6		2	1	c				1	1	2			
7×6													1			1		4		5		c					2	1	1		
6×8																					1	11		c					1		
8×6																					1	4	1	c		4	5		2		
6×9																	1	1		3		1	2			c	9	10	2	3	
9×6																	1			11		3	2		2	c	10	2			
7×7												1						1				6			c						
7×8																						6		7		4	c	2	1	3	
8×7																						4		4		6	c		1	2	
7×9																				1		2	1	1	9	3	12	c		2	
9×7															2					3		1				4	8	c	1	5	
8×8																				3				1			1	2	c		7
8×9															1							1				1	4	4		c	3
9×8																								1		1	1			c	3
9×9																				1											c

Table 2.1. Error matrix for 60 subjects (Campbell & Graham 1985, appendix A). Each element represents the number of times a product occurred as an incorrect answer to the corresponding problem (c = correct). Given a total of 8640 trials (each subject tested on 4 blocks of 36 problems), and an error rate of 7.65 per cent, it appears that each "c" in the table corresponds to an average of 125 correct recalls. In addition to the errors shown here, Campbell & Graham labelled 17 errors above 5×4 as involving products less than 20.

- Frequent product errors, where the error is one of the five products 12, 16, 18, 24 or 36. These products happen to occur more frequently than most in the multiplication

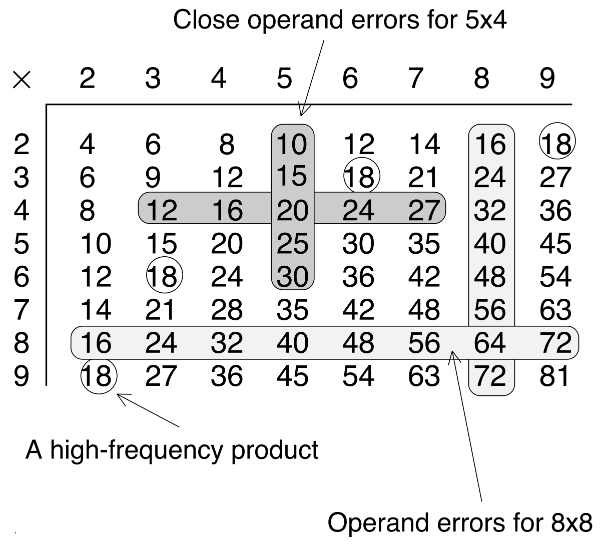

Figure 2.2. Slip categories. Note that 4×3=13 is a non-table error because the number 13 does not appear as a product in this table.

table defined over integer operands from 2 to 9.

- Table errors, where the erroneous product is the correct answer to some problem in the range 2×2 to 9×9, but the problem does not share any digits with the presented problem (e.g., 4×5=12).

- Operation errors, where the error to $a \times b$ is correct for $a + b$. These errors occur even in blocks of problems which only contain multiplication problems. Hence, it is unlikely that they are only the result of a perceptual slip (Winkerlman & Schmidt 1974).

- Non-table errors, when the answer is not a product found in the multiplication table. For example, 4×3=13 is a non-table error because 13 is not a number found in the multiplication table defined over 0×0 to 9×9. In the Campbell & Graham study, only 7.4 per cent of errors were in this category, and they are not shown in table 2.1.

The main error categories are summarized in figure 2.2.

Table 2.2 summarizes the distribution of errors shown in table 2.1. For comparison, the results from a similar study by Harley (1991) are also shown. The statistics for the Harley

	Campbell & Graham	Harley
Operand errors	79.1	86.2
Close operand errors	76.8	76.74
Frequent product errors	24.2	23.26
Table errors	13.5	13.8
Operation error	1.7	13.72
Error frequency	7.65	6.3

Table 2.2. Percentage breakdown of errors. Figures are mean values for sixty adults tested on 2×2 to 9×9 from Campbell & Graham (1985, appendix A), and 42 adults tested on 0×0 to 9×9 from Harley (1991, appendix B). For the Campbell & Graham data, the operand error and operation error percentages are an approximation due to incomplete data.

column, apart from error frequency, were recomputed from Harley (1991, appendix B). As can be seen from the table, most errors are operand errors, and more specifically, close operand errors. Both studies agree on this, but note the difference in the operation errors row. Harley's study included the 0×0 to 9×9 problems for which there are numerous errors of the form 0×N=N. Approximately 11 per cent of the errors were 0×N=N errors, and these have been classified as operation confusion errors in table 2.2.

The RT and errors are tied together by Campbell's (1987, p. 110) observation that there is a strong correlation of 0.93 between error rate and RT. That is, the problems on which a subject is slowest are also the problems that the subject often gets wrong.

2.1.2 *Neuropsychological constraints*

Any good model of a cognitive skill should be structured in a way that is similar to the equivalent human system. Assuming that brain damage can selectively impair parts of this structure, then it is reasonable to expect that the model can be damaged to behave in ways that brain-damaged individuals do. That is, results from brain-damaged subjects suggest the question: "What must the normal system be like in order that, subsequent to selective damage, performance breaks down in just the ways observed?" (Sokol et al. 1991, p. 355).

Results from brain-damage studies (e.g., Sokol et al. 1991; McCloskey, Aliminosa & Sokol 1991) offer further constraints on models, in addition to the empirical findings reported above.

Brain damaged subjects, unlike normals, make a number of omission errors, responding to problems with “don’t know”. For example, McCloskey, Aliminosa & Sokol (1991, p. 177) report that two subjects failed to produce answers for 24 and 46 per cent of trials. On the trials where the subjects did produce incorrect answers, there was a tendency for larger problems to show greater impairment than smaller problems. However, this impairment was non-uniform. For example, one subject (SB) had an error rate of 63 per cent on 8×3 and 7×4, but was correct when answering 8×6 and 7×6. The errors made tended to follow the distribution discussed above: most of the errors were close operand errors.

2.1.3 Rule based processing

Figure 2.3 shows the RT for problems 0×0 to 9×9. There are persistent statements in the literature that zero and one times tables are governed by procedural rules. Evidence cited to support this notion comes mostly from the relative RTs of zero problems. See, for example, Campbell & Graham (1985, p. 341); Miller et al. (1984, p. 51); Stazyk, Ashcraft & Hamann (1982, p. 334).

However, the most persuasive arguments for a separate mechanism for zeros and ones problems comes from studies involving brain-damaged subjects. Sokol et al. (1991) reports that one patient (PS) showed errors of 0×N=N on almost all 0×N problems, and then suddenly improved to almost perfect performance on all the zeros problems (p. 358). That is, unlike other problems, zero problems show uniform impairment.

Further support comes from remediation studies (McCloskey, Aliminosa & Sokol 1991, p. 186–7). Patient JG was trained on the problems 0×8, 0×9, and some non-zero problems, including 2×7. Although training on 2×7 leads to improvement on 7×2, JG was correct on only 5 out of 355 problems that were not trained on. In stark contrast is the generalization on zero problems: after training on two zeros problems, JG was 99.3 per cent correct on all other untrained zero problems (139 of 140 problems correct). Similar remediation results suggest that a rule N×1=N is also utilized.

Despite the convincing evidence for a rule-based mechanism, it is not clear why certain errors occur for zero problems. The majority of errors for zero problems take the form of $0\times N=N$ (Harley 1991). McCloskey, Aliminosa & Sokol suggest two explanations for why these errors occur.

The first possibility is that the wrong rule is retrieved. That is, when presented with a $0\times N$ problem, the subject attempts to retrieve the rule $0\times N=0$ but actually finds $0+N=N$ or $1\times N=N$. However, it is not clear why $0\times N=N$ errors occur for the problems 0×1 and 1×0. In these cases, the rule $1\times N=N$ should be applied to give the correct answer.

Another possible source of $0\times N=N$ errors could come from stored incorrect rules. That is, the subject stores a rule like $0\times N=N$ alongside the correct rule. McCloskey, Aliminosa & Sokol offer a scenario for how such rules are acquired: the incorrect rule $0\times N=N$ is a generalization of the rule learned from addition, $0+N=N$. The generalization is: any operation involving zero and N results in N. Ordinarily the correct rule will be used either because it was the most recently learned rule, or because it is the most specific rule. The problem then is how brain-damage and normal forgetting can lead to the selection of the incorrect rule in preference to the correct rule. One account is that "…the correct rule [$0\times N=0$] seems essentially arbitrary to many individuals, where as the incorrect rule seems to fit better with other knowledge of arithmetic. In other words, to some individuals the incorrect rule may make more sense than the correct rule" (McCloskey, Aliminosa & Sokol 1991, p. 192).

McCloskey, Aliminosa & Sokol conclude that, although the account of rule-based arithmetic is speculative, arithmetic recall is characterized as an associate network of facts for 2–9 problems augmented, with rules used for zero and ones problems. However, the problem is that no models have been produced which incorporate rule-based processing together with recall from an associative network. Such a system needs to be built to articulate the details of rule-based recall. Also, it is not yet clear exactly what constitutes a rule (Kirsh 1990; Clark 1993). Given these uncertainties, it may be worth first pursuing single mechanisms models to account for the rule-like behaviour. This is further discussed in section 3.5.2.

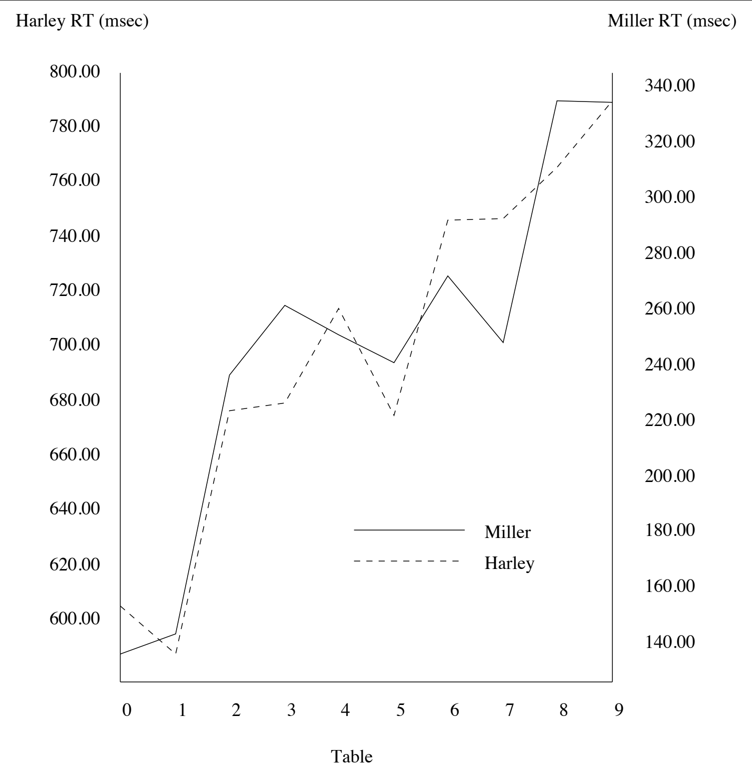

Figure 2.3. Plot of mean correct RT per multiplication table collapsed over operand order for: median RT of 42 adults (Harley 1991, appendix D); median RT (adjusted for naming time) of 6 adults (Miller et al. 1984, table A1).

2.1.4 Summary

The major findings which constrain a model of arithmetic recall are:

1. The problem-size effect indicates that problems with large operands tend to take longer to answer than problems with small operands.

2. There are exceptions to the problem-size effect, most notably the 5 times table and tie problems.

3. The majority of errors are operand errors, where the incorrect answer is correct for a problem that shares an operand with the presented problem.

4. Recall, when damaged, shows a pattern of non-uniform damage, rather than a global degradation in performance.

5. Zero and ones problems appear to be rule-governed.

Arithmetic is a well learned skill: a large number of individuals are taught and "know" the times tables. Yet some problems are easier than others, and people do make occasional mistakes when asked to recall these supposedly well-learned facts. What mechanisms capture the time-course of this skill, and how are errors produced? What kind of learning process installs these mechanisms?

2.2 *Previous models*

Various approaches have been taken to modelling the above phenomena. Examples included: counting models, sometimes called digital models; analogue models; network models; and procedural models, based around rules for arithmetic.

Counting models assume that adults have internalized children's counting strategies. For example, one of the most successful counting models is the MIN model (Groen & Parkman 1972). To add two numbers, n and m, two counters are postulated: the first is set equal to the larger of the two numbers (say, m); the second is used to increment the first counter n times. Hence, addition is achieved by repeated incrementing, and the RT will be a function of the smaller (MIN) number. Although this model is a good predictor of addition RT, there are problems when the system is applied to multiplication.

No counting account has been proposed for multiplication, but as Stazyk et al. (1982) note, there are some straightforward ways to extend the model to multiplication. The simplest scheme assumes that the first counter is set to zero, and then incremented by the larger number, m, for a total of n times. However, this proposal requires that the subject be able to count not only in 1s, but also 2s, 3s, etc, and that the size of the increment should not effect the speed of counting. Ashcroft & Stazyk (1981, p. 47) note:

> Since the counting model proposed for addition asserts that subjects count in units of one, the suggestion that adults can count in units of varying size when multiplying, but not when adding, lacks internal consistency.

In addition, confusions between addition and multiplication, namely operand errors, also suggests a single system for both skills (Winkerlman & Schmidt 1974; Miller et al. 1984), and this is something difficult to capture in a counting system. One of the reasons for postulating the MIN model was to capture the problem-size effect. However,

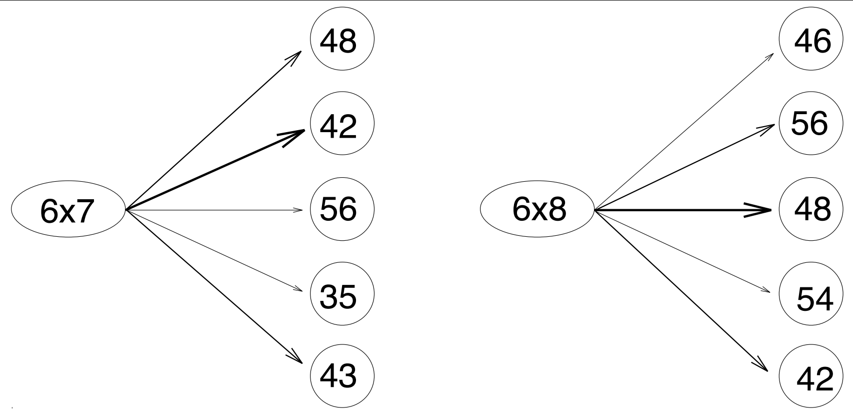

Figure 2.4. The problem-to-answer representations for the distribution of associations model. Line thickness depicts strength of association (figure based on figure 3 from McCloskey, Harley & Sokol 1991).

the counting model has difficulties with the 5s and ties exceptions to the problem-size effect.

Little progress has since been made with counting models, and no detailed models of errors have been proposed. Attention has been focussed on network models, and these are discussed in this chapter (see Cornet, Seron, Deloche & Lories 1988, for a detailed review of counting models and other approaches).

The set of network models include the most explicit models that exist. Unlike the reconstruction of answers performed by counting models, network models assume an associative memory structure. Activation is assumed to spreading between problems and answers nodes. An review of network models is given by McCloskey, Harley & Sokol (1991). Two of these models are presented next.

2.2.1 *Distribution of associations*

One of the more interesting network models is the distribution of associations (DOA) model proposed by Siegler (Siegler 1988, 1987; Siegler & Shrager 1984). The model is an account of strategy choice—why retrieval is used sometimes, and other methods, such as repeated addition, are used on other occasions. The network of associations that make up an adult's multiplication memory is formed by the success and failures of children's back-up strategies.

Each problem is connected to a set of candidate answers consisting of the correct product plus some incorrect answers. For adults it is assumed that the strongest associations are to correct products (see figure 2.4). For children, however, the distribution of associations will be "flatter", representing limited experience with multiplication.

The recall process begins by randomly selecting a confidence criterion. Siegler (1988) notes that "…even 1-year-olds often will not state an answer that is suggested to them if they do not believe the answer is correct. Such resistance suggests that they possess a standard, akin to a confidence criterion, for stating an answer" (p. 261).

An answer for a presented problem is retrieved according to the strength of the association between the problem and the answer: the stronger the association, the more chance the answer has of being selected. If the association to the proposed answer is stronger than the confidence criterion, the answer is stated. In the cases where the strength falls below the confidence criterion, another answer is selected. This loop continues until a maximum number of attempts has been made (1, 2 or 3—a number randomly selected at the start of the retrieval process).

If no answer has been stated after the maximum number of attempts, one of two processes can, probabilistically, follow: sophisticated guessing, or problem elaboration.

Sophisticated guessing is just another attempt at retrieval from the associative network. However, no comparison is made to the confidence criterion: the answer is immediately stated.

For elaboration, the model "writes down" the problem. For human subjects, this is done with pencil and paper. For the computer model, this is implemented by temporarily boosting the strength of the associations by 5 per cent. Another retrieval attempt is then made, and the answer stated if it exceeds the confidence criterion—and this is more likely with newly increased strengths. Siegler (1988) comments: "The written elaboration may itself be associated with possible answers to the problem" (p. 261). It is not clear why this elaboration would help given the modular structure of arithmetic skills outlined in chapter 1. That structure indicates that presentation form (visual Arabic numerals or verbal) is independent of the arithmetic fact store. Nor is it obvious how increasing associative strengths is equivalent to the writing down of a problem.

If no answer is given after elaboration, a back-up strategy is used. The strategy presented by Siegler is repeated addition, where a counter is incrementing by n, m times for the problem $n \times m$.

Reaction time

RT is dependent on how many of the above stages were processed before an answer was produced. Problems produced by repeated counting have to have passed through the recall stage and either elaboration or sophisticated guessing. Siegler (1988, p.272) reports that the RTs produced by the model can match those of children and adults. The cause of this is outlined below.

Errors

Errors can arise from either: retrieving an incorrect answer that happens to exceed the confidence criterion; or from miscounting when using a back-up strategy. For miscounting, there is a fixed probability that a number is added twice or skipped over. Also, two numbers could be added incorrectly. For example, on 2×8, the model could produce $8 + 8 = 15$. The frequency of these errors is proportional to the size of the number being added, except for 5s problems which are added as accurately as 2s. Siegler reports that children are particularly accurate at adding 5s, hence the exception.

Learning

Every time an answer is produced, the association between the problem and answer is increased. Due to (presumed) external reinforcement, the increment is twice as much for correct answers than incorrect answers.

Initially, the associative strengths are small (0.01), so retrieval is likely to fail, and the repeated counting strategy would be used. The correct association will form after the problem had been presented a number of times assuming that repeated counting usually produces the correct answer. As associations grow in strength, recall will succeed more often, producing more correct answers in a shorter RT. Hence, the model accounts for the development towards relying on recall and for RTs to decrease.

The problem-size effect is derived from the assumption that repeated counting is more prone to errors on larger problems. This is seems likely because more addition steps are needed for larger problems. So for larger problems, repeated counting will produce a distribution of associations that is "flatter" for large problems than small, resulting in longer RTs and more errors. Exceptions to the problem-size effect can be incorporated into the model. An example is the 5s problems discussed above for which the back-up strategy reliably produces the correct answer, meaning that strong associations will quickly form for these problems.

Problems are also presented according to the frequencies found in a school textbook. As smaller problems tend to be presented more often than larger problems, this adds to the effect (discussed further in section 3.3.1).

Discussion

Siegler's DOA model is based on an associative network in which problem nodes are linked to candidate answers. "Associations across arithmetic operations (e.g., addition and multiplication) are also hypothesized to be present in the network" (p. 259), but this is not discussed in any detail by Siegler.

Operand errors are accounted for by the way the back-up strategy miscounts. When adding a group of 6s for the problem 4×6, the strategy may add one too many, resulting in 30, or one too few, 18. Both are operand errors. Moreover, because the strategy is more likely to err once, rather than twice or more, operand errors will be close to the correct product.

The other way in which the back-up strategy can produce an incorrect answer is by adding two numbers and getting the wrong result (e.g., $2 + 2 = 5$). It turns out, from McCloskey, Harley & Sokol's (1991) analysis, that this behaviour will produce more non-table errors than table errors. This is the opposite of the results reported by Campbell & Graham (1985) for adults. Siegler presumes that counting errors will be distributed around the correct answer. That is, the error 6×7=41 (under counted by 1) and 6×7=43 (over counted by 1) occur more often than errors that result from larger miscounts. Given this assumption, McCloskey, Harley & Sokol note that for the 64 problems 2×2 to 9×9, errors of the form $(n \times m) \pm 1$ turn out to be non-table errors twice as often as table errors. Hence, Siegler's model predicts the opposite of the observed situation, where table errors occur twice as often as non-table errors for adults.

Although more explicit than many models, there are some details missing from the DOA account. For example, answers are retrieved from the associative network with a probability proportional to associative strength. This aspect of the model is not fully elaborated: is it a spreading activation system? If so, how does activation spread? By what mechanism are responses selected? As will be shown later, many models have focused on just this aspect of retrieval.

The model contains a number of separate mechanisms: recall, elaboration, sophisticated guessing, and back-up. No details were given about the nature of the overall control mechanisms: what factors determine why sophisticated guessing is used sometimes (20 per cent of trials), but elaboration used at other times? How and why is the associative strength temporarily boosted for the elaboration stage?

The simulation results reported by Siegler (1988) were for just 20 problems due to limited resources, and this should be extended.

Finally, it is assumed that back-up strategies behave in the ways specified. It would be interesting to see a more detailed account of the structure of these strategies, explaining why they err in the ways that they do.

The DOA model is an interesting account of strategy choice. As a model of recall of multiplication facts, it needs to be further specified.

2.2.2 Network interference

Campbell & Graham's (1985) "network interference" (NI) model does not concern itself with multiple strategies as the Siegler model does. Rather, just the associative retrieval component is considered.

Figure 2.5 shows the architecture of the model, which consists of four kinds of units:

1. Operand units, to encode the 6 and 7 of 6×7.

2. Problem units, activate only for specific problems.

3. Product units.

4. General magnitude units.

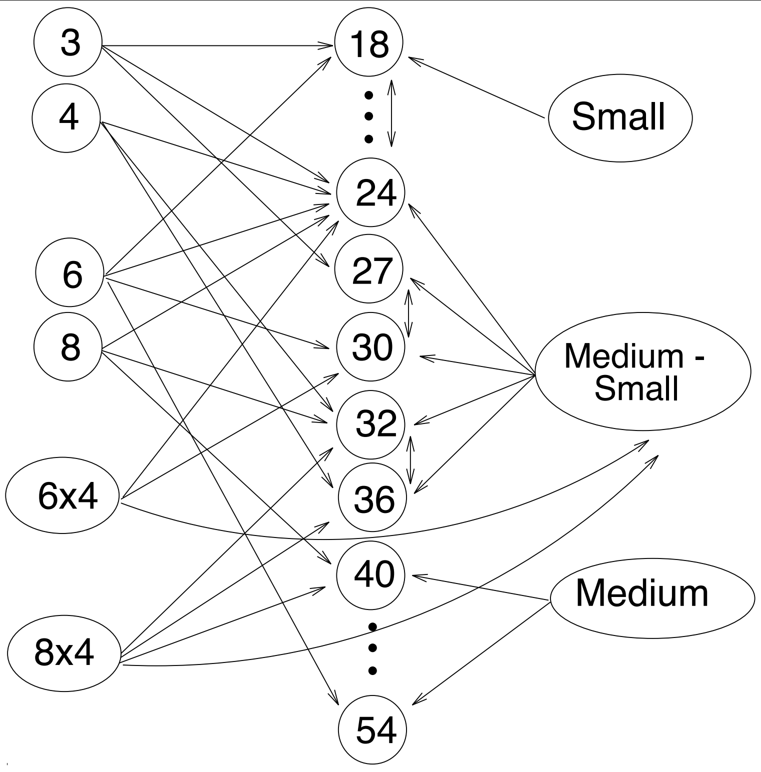

Figure 2.5. Network interference model. Based on figure from McCloskey, Harley & Sokol (1991).

Operand units are connected to the products in the operand's table. That is, the operand unit 3, for example, is connected to the products 6, 9, 12, 15, and so on. "Type" product units are used—there is only one product unit representing 24, which is activated by 4×6 and 3×8. The alternative to type product units is token units, in which there would be two 24 units, one activated by 4×6 and the other by 3×8.

To disambiguate certain answers, a set of problem units are used. For example, with type product units, there is no way to choose between 32 and 24 when 8 and 4 are presented to the system: 32 is doubly activated by the 4 and 8 in 4×8; but 24 is also activated, once by the 8 (3×8=24), and again by the 4 (4×6=24). With problem units, the correct product will receive an extra input.

Problem units also activate the magnitude units, which in turn activate an appropriate set (large, medium, etc.) of product units. This allows the subject to estimate the size of the answer. Product units that have answer digits in common—such as 12 and 15, or 12 and 32—are connected together.

Learning

The NI model suggests that problem presentation frequency and order determine the strengths of the associations in the network. Campbell & Graham (1985) assume that smaller problems are learned first, and are presented more frequently, than larger problems. These assumptions are not unreasonable, and chapter 3 looks in detail at presentation frequencies. However, for problem frequency there is an inherent difficulty in discovering the actual frequency experienced by subjects. Siegler's results are based around the frequency of problems from school textbooks, but children certainly experience arithmetic problems other than those found in textbooks.

Nevertheless, given the assumptions stated above, Campbell & Graham predict that the first problems learned will impair the learning of later problems (proactive interference). These smaller learned problems will also act as a source of error for larger problems. That is, during learning, the smaller problems will not have many alternative answers to produce as an error.

Errors

Operand errors may occur because each operand unit is connected to a set of product units. On some occasions, the wrong product may be selected, giving an operand error. Close operand errors are produced because the general magnitude units activate products that are approximately the correct magnitude for the presented problem.

Errors also occur because of false associations. During learning, inputs are associated with the correct product. However, due to the fact that activation spreads, other product units will be active. Associations will also be made to these, slightly active, false products.

Learning usually occurs in lessons, where a number of problems will be presented and solved. Campbell & Graham argue that residual activation of problem units and product units other than the one being solved will be associated with the current problem. In this way, false associations are formed, which may be produced as errors. In particular, it is assumed that problems close in magnitude (e.g., 7×8 and 7×9) are more likely to be learned in the same lesson than more distant problems (such as 7×8 and 7×2). As McCloskey, Harley & Sokol (1991) note, this suggestion is not backed by any empirical evidence. If true, it would further contribute towards the production of close operand errors.

The connections between products that share digits will produce some table errors, but this seems rather "ad hoc—the associations between answers sharing a digit seem to play no role in the model other than that of explaining certain table errors" (McCloskey, Harley & Sokol 1991, p. 393).

As all the answers units correspond to a product, there is no account of non-table errors.

Retrieval

One of the founding ideas behind the NI model is that the activation of incorrect answers interferes with recall of the correct answer. That is, as interference increases, RT will become longer, and there will be an increased chance of an error.

As mentioned, retrieval is a spreading activation process. The rules governing the spread of activity are not specified, and nor is the response mechanism. Presumably the most active product unit is selected as the answer. RT is assumed to be a function of answer activity, but again, this is not specified.

Discussion

The lack of specific detail with this model (no activation rules, no response mechanisms for correct or erroneous answers) is not surprising given that the system was not implemented. As it stands the system is a conceptual analysis of arithmetic and raises a number of questions:

- Why so many kinds of nodes? What does each knowledge source contribute to the model? Are they all necessary?

- How do these knowledge sources develop? In particular, how are the magnitude units formed?

- How are problem-size exceptions handled? Although the system can accommodate exceptions, it would presumably have to be via problem frequency or order. Campbell & Graham (1985) briefly comment on the possibility of a rule-based system for 5s (the product should end in zero or five), but do not pursue it in any depth.

- By what means are the associations strengthened? A brief description was given, but without more specific learning mechanisms it is not possible to determine the relative importance of each kind of link.

The model itself lacks detailed mechanisms, but provides a useful source of ideas for what factors may contribute to the phenomena.

2.3 *Previous connectionist models*

The rest of this chapter considers those models that are strongly connectionist. It seems apparent that connectionism is appropriate for this task, and that the rigour of implementation could flesh out some of the modelling issues introduced above.

The first connectionist account was built and developed by a group at Brown University from 1989 onwards (J. A. Anderson, Spoehr & Bennett 1991; Viscuso, Anderson & Spoehr 1989; Viscuso 1989; J. A. Anderson, Rossen, Viscuso & Sereno 1990). This account is unsatisfactory for reasons set out in section 2.3.1. Graham (1990) also performed some experiments with a connectionist network, but the model lacks an explicit recall mechanism.

No less than three other connectionist models were built in answer to McCloskey, Harley & Sokol's call to "shift from a demonstration of the framework's basic merit to the hammering out of detailed, fully elaborated models" (1991, p. 394). These models were developed independently at more-or-less the same time. The first to be published was the model by McCloskey & Lindermann (1992), followed by the model described in chapter 3 (Dallaway 1992a, 1992b). The most recent model was built by Rickard, Mozer & Bourne (1992). The connectionist models are described below, and compared in the next chapter.

2.3.1 *Brain-state-in-a-box*

The brain-state-in-a-box (BSB) model (J. A. Anderson, Spoehr & Bennett 1991; Viscuso et al. 1989; Viscuso 1989; J. A. Anderson, Rossen, Viscuso & Sereno 1990) has been presented in a number of forms. Initially it was built for "qualitative multiplication", in which answers are only "ballpark estimates". That is, the model had a small number

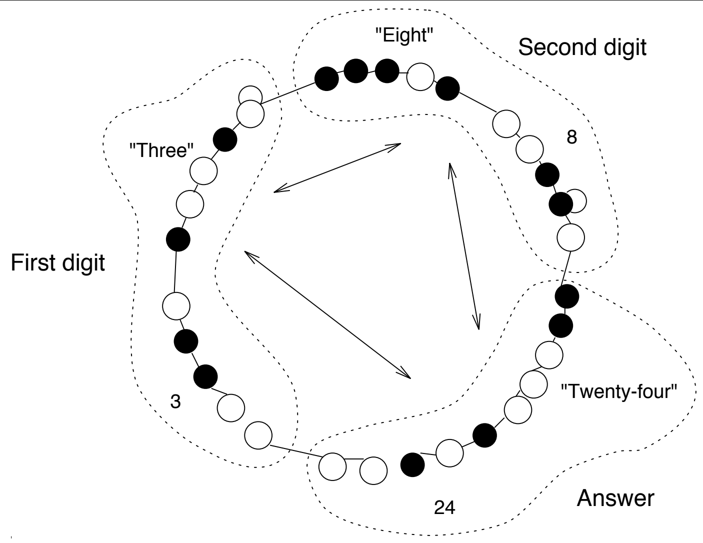

Figure 2.6. BSB network. This figure is a simplified picture of the 1266 unit network described by J. A. Anderson, Spoehr & Bennett (1991).

of answers (e.g., 3×1=5, 3×2=5, 6×2=10). Most of the BSB simulations are of qualitative multiplication, but J. A. Anderson, Spoehr & Bennett (1991) have started to model quantitative multiplication.

The most recent simulations present a network with 1266 units. The units are split into three groups (see figure 2.6): two sets of units for the two operands, and another for the answer. Each of these three groups consists of two sets of units: a bar encoding of the number, and an arbitrary symbolic label.

The bar encoding is the activation of a consecutive number of units. That is, a number is encoded by the positioning of the bar along the set of units, forming an "analogue" encoding. The units are arranged in an approximately logarithmic fashion, so that large numbers are less distinct from each other than smaller numbers.

Originally, the symbolic component encoded the name of a number (e.g., "one") based on the ASCII representation of characters. McCloskey & Lindermann (1992) noted that this representation is unmotivated. Problems are usually encountered as Arabic numbers or spoken works, but almost never as written words, such as "eight times six". Yet, it was just this spelling of the numbers that was used as input to the network. It now seems that the symbolic input is no longer supposed to represent the spelling of a number, and is just presented as a random vector. This input still remains unmotivated, and it is not clear what purpose it serves in accounting for the phenomena.

For each operand and answer there seems to be 150 units to represent the symbolic and bar part. From the reports on the system, it is not clear how the units are divided up for each input field, or how wide the bar is. This uncertainty is also commented upon by McCloskey & Lindermann (1992).

The large number of units means that simulations require a large amount of CPU time. Because of this, only 32–34 facts are trained. In an attempt to scale the system to all the multiplication problems, an extra field was added to the answer field. This new field contains 60 units to encode nine "integer symbols" and "…tries to capture hazy intuitions about two digit numbers having a most and least significant digit" (J. A. Anderson, Spoehr & Bennett 1991, p. 7). No further details were given about this new field: does it represent the "tens" or "units" part of the answer? How does it affect recall?.

For training, the operand and answer units are clamped with desired activations, and the weights between the units are adjusted using the Widrow-Hoff error correction rule. In previous simulations (Viscuso et al. 1989), each problem was presented 30 times, in a random order.

Recall

When simulating recall, the operand units are clamped, and the answer units compute their activation as follows:

$$o_i(t+1) = \alpha \sum_j w_{ij} o_j(t) + \gamma o_i(t) + \delta f_i(0)$$

where: $o_i(t)$ is the output of unit i at time t, which is limited to be between -1 and $+1$; α is the feedback constant; γ is the decay constant; and $f_i(0)$ is the external input to the unit, multiplied by a constant δ.

The state of the system representing associations between operands and answers forms an attractor. The units in the network continue to update until the system arrives at one of these point attractors. Cycling continued until "all of the vector elements were saturated" (Viscuso et al. 1989, p. 150). That is, there was a read out threshold, and once all the units had exceeded this threshold, processing stopped. The number of cycles required to reach this point is taken as the RT for the presented problem. On some runs updating was stopped when a maximum number of cycles (60) was reached.

Reaction time

J. A. Anderson, Spoehr & Bennett (1991) report that a number of their BSB simulations exhibit a problem-size effect. The RT for 32 reported problems do seems to follow the problem-size effect. For this small sample of problems it is not clear if there are exceptions for 5s, or if tie problems are accounted for.

J. A. Anderson, Spoehr & Bennett agree with Campbell & Graham that problem frequency and order are important causes of the problem-size effect. It does not appear that the BSB simulations make use of problem order or frequency, although it is noted that: "Practice and order effects are, of course, easy to model within an associative framework" (p. 13).

It is further suggested that the problem representation is also responsible for the problem-size effect: "…the codings for problems of increasing magnitudes may get successively more 'muddled together'". However, "…it does not appear that the problem-size effect resulting solely from the bar code compression is a particularly robust feature of the modeling" (p. 13).

Verification

With the architecture of the BSB model it is possible to study the verification task. Distant false problems (such as "5×2=60?") and close false problems ("5×2=20?") were presented to the network by clamping both operand and answer units. If, during processing, the network overwrites the answer field, the problem is considered rejected. If the answer field remains unchanged, the problem is accepted. Simulations showed that the network would reject distant products faster than products that were close to the correct product. The system also accepted near false products, but no distant false products.

Errors

Adults are quite capable of learning the multiplication facts, although with some difficulty. Any errors are, as McCloskey & Lindermann (1992) calls them, occasional: errors in retrieval, and not predominately errors resulting from false beliefs. This is not the case for the BSB model. The network, as presented, is simply incapable of learning all the associations. For example, on 8×4 the network produced the answer 24 or 28, but never 32. This runs against the notion that slips are one-off run-time errors, rather than permanent disabilities. Correct answers are reconstructed for about 70 per cent of the problems.

The errors that are made are close in magnitude to the correct answer. Examination of the results given in J. A. Anderson, Spoehr & Bennett (1991) shows that some of the errors are close-operand errors, others are non-table errors. No details are given on the proportions of the errors generally, or how non-close operand, table or operation errors occur.

Discussion

Although there have been a number of papers describing the BSB model, details of the model are still unexplained, and only small scale simulations have been performed. A number of points need to be clarified:

1. The motivation for the symbolic component of the representation, and the role it plays in explaining the phenomena, is uncertain.

2. The details of the input representations (e.g., the "roughly logarithmic" operand units) need to be given.

3. The 60 units used to encode "integer symbols" in the answer field remain a mystery. What do they represent? What is the motivation for it?

4. There is no account of occasional, run-time, errors.

5. The RT results do not cover ties, or the other exception to the problem-size effect.

6. The results from qualitative simulations are interesting, but what bearing does this have on the multiplication done by people?

McCloskey, Harley & Sokol (1991) comment that the Viscuso et al. (1989) "proposal has several limitations and cannot be considered a well-articulated model", but add that "the [neural net] approach probably merits further exploration" (p. 395).

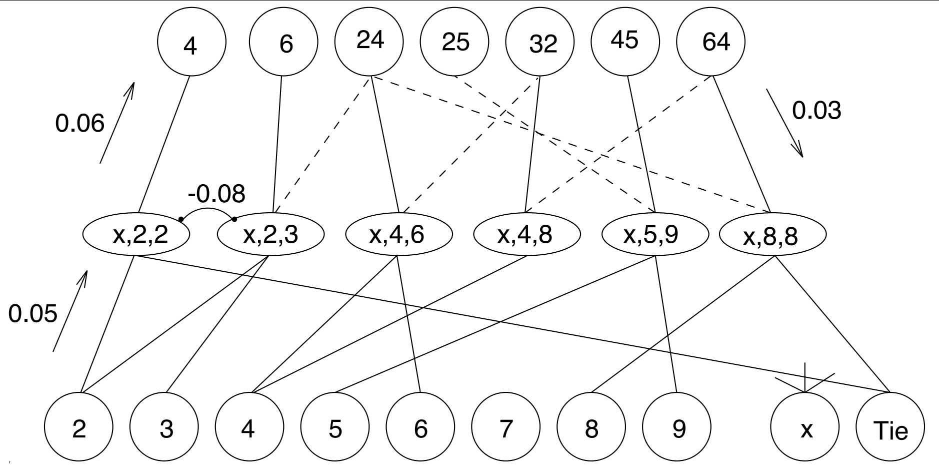

Figure 2.7. Interactive activation network (from Rickard et al. 1992, figure 5).

2.3.2 *Backpropagation*

Graham (1990) has run a large number of simulations looking at how a connectionist network can store arithmetic facts. His model contained two sets of input units to encode the two operands connected to a set of hidden units. The hidden units were connected to a set of output units divided into tens and units fields. Various parameters were explored, including: number of hidden units; learning rates; input and output encodings; problem presentation ordering; and rehearsal schedules detailing when and how often trained problems were re-presented.

A problem-size effect was found by training on small problems first. The problems 2×2 to 9×9 were split between a large group (products greater than 32) and a small group. Training with backpropagation on the small group first produced a problem-size effect for the accuracy of the network. That is, more smaller problems were correct than larger problems.

Errors were measured by testing the network at various stages during learning, at 20 per cent correct and 70 per cent correct. These errors were mainly operand errors.

Graham's work cannot be considered as a model of the recall process because it provides no explicit retrieval processes by which errors and RTs can be measured. Nevertheless, it is a useful study, and suggests that connectionist system may be useful in this domain.

2.3.3 *Interactive activation*

Like the BSB model, the interactive activation (IA) model proposed by Rickard et al. (1992) is a settling network. The architecture of the network is shown in figure 2.7.

There is just one set of input units to represent both operands. Each operand is connected, by a fixed weight, to every problem unit which contains that operand. The problem level is motivated by the success of other associative memory models that use a part-whole hierarchy—such as the interactive activation model of word perception, (McClelland & Rumelhart 1988, chapter 7). At the problem level there is mutual inhibition (again, by a fixed value) to implement the constraint that only one problem unit should be active. A set of type product units receive activation from, and send activation back to, the problem units. There are also false associations between problem units and product units. All the weights in the network are hand-crafted; learning is not accounted for.

Processing begins when a problem is presented and the appropriate operand units are clamping on. A multiplication unit is also activated for all problems, and presumably future extensions will include units for addition, subtraction and division, too. The correct problem unit will receive input from three nodes (the two active operands and the multiplication unit), and table-related problem units will receive input from just two nodes (one of the operand units and the multiplication unit). Hence, the correct problem unit will be most active, inhibiting other units, although table-related problems will be slightly activated.

Note that for tie problems only one operand unit would be activated. This means that tie problem units will only be activated by two inputs, as will all the other problem units associated with the operand. In order for the correct problem unit to be activated, Rickard et al. include a "tie" unit as part of the input. This unit is activated whenever a tie problem is presented, supplying the tie's problem unit with three inputs. The motivation for the ties unit stems the notion that there is an "…intrinsic uniqueness in the representations/processes…that underlie performance on these [tie] problems" (Rickard et al. 1992, p. 18). That is, there is "something special" about tie problems, and this is represented by an extra bit on the input layer.

All the units in the IA network are updated synchronously. The activation of a unit is just the net input to the unit, but is never allowed to drop below 0 or go above 1. Processing stops when a product unit exceeds a threshold of 0.8.

As described so far, the system will always produce the correct response. Rickard et al. report that an incorrect product units never exceed an activation of 0.1. RT is the number of cycles it takes before a product unit exceeds the threshold. All problems reach threshold at the same time (after 43 cycles).

Response times

To explain the problem-size effect, Rickard et al. hand modified the problem-to-product weights. It is assumed that the more frequently occurring problems will cause stronger associations between units. To implement this, the problem set was split into two sets: "small" problems, with products of 30 or less; and the remaining "large" problems. Based on the assumption that larger problems are experienced less often, the weights between all units involved in the larger problems were decremented by 10 per cent from the values given in figure 2.7. After this change 42 cycles were required for small problems, and 51 for the large problems.

False associations are also assumed to interfere with retrieval. To simulate this, false connections were made between a problem and various product units—one each for an operand error, close operand error, table error, and non-table error. The weights for the incorrect associations were set at 0.025 for problem-to-product links, and 0.0125 for product-to-problem. With the model modified in this way (independently of the changes described above), cycle time increased to from 43 to 48 cycles. Again, assuming that larger problems have stronger false associations, Rickard et al. state that false associations can contribute to the problem-size effect.

Errors

Errors are only accounted for implicitly. It is assumed that the probability of an error occurring is proportional to the activation of the product during cycling. As described above, the products associated with operand errors and close operand errors are more active during processing than table-unrelated products. Hence, given that more active products are more likely to be errors, the IA model predicts more frequent operand errors than table-unrelated errors.

Primed production and verification

The primed production task is simulated by activating the product node corresponding to the prime, and allowing the network to cycle 6 times to simulate brief exposure. Activation is then removed from the primed node, and retrieval begins as described above. When the correct node is primed, the system's response is faster: 38 cycles compared to 43 without priming. The IA system slows when an incorrect prime is presented, and slows more when the incorrect prime is related to the correct answer. When false associations are included this increase in RT is even more prominent. These results are consistent with those reported for human subjects.

Similar results are reported for the verification task. The verification task is simulated like primed production, but leaving the candidate answer mildly activated (external activation of 0.01) for the duration of the experiment. False problems slow the system, and this varies with the degree of similarity between the correct and primed product. Note that the system is not changed in any way to output "yes" or "no" in answer to the verification task. The answer is selected by the mechanism described above. Presumably, the decision as to the whether the candidate product is correct or not, is a stage to be added after the production of a product.

Rickard et al. (1992) also introduce a verification strategy based on "resonance" as an alternative to the above method. Resonance is defined as a global measure derived from Smolensky's (1986) "harmony" measure:

$$\text{harmony} = \sum_{ij} o_i o_j w_{ij} + \sum_{i} \text{ext}_i o_i$$

where o_i is the output of unit i, w_{ij} is the weight between units i and j, and ext_i is the external input to unit i. The harmony, or resonance, of the system, as defined here, will increase as the "self-consistency" of the network increases. That is, the correlations between active units connected by strong positive weights increases. During verification tasks, Rickard et al. show that harmony does increase across cycles, and it increases faster with a correct candidate. Harmony increases slowly when an operand error candidate is presented, but increases more quickly as the table-relatedness of the candidate is reduced to table-unrelated candidates. “Thus, under the assumption that there is a criterion resonance for responding ‘true’, the probability of responding ‘true’ incorrectly based on resonance in memory is greater when a candidate answer is table-related and/or has an incorrect association to the problem representation” (Rickard et al. 1992, p. 28). There is no discussion of how the global resonance measure is computed by the system.

Discussion

The IA model has a general problem: there are too many assumptions, and not enough quantitative simulations. Indeed, it is not possible for Rickard et al. to produce many simulations without specifying some more recall mechanisms. In particular, there needs to be an explicit account of error production. As J. R. Anderson (1983, p. 88) notes:

> Activation does not directly result in behaviour. At any point in time a great deal of information may be active…There must be processes that convert this activation into behaviour. A serious gap in much of the theorizing about spreading activation is that these processes have not been specified.

This criticism applies directly to the IA model. It is not enough to assume certain kinds of associations will form, or that error probability is proportional to activation. It was already clear that associative networks of many kinds (e.g., Graham 1990; Viscuso et al. 1989) could be constructed to account for some of the empirical observations on human behaviour. As McCloskey, Harley & Sokol (1991) have commented, network models show a great deal of promise for this domain, but “detailed, fully elaborated models” are required. The IA model lacks these details:

1. There is no learning mechanism. As discussed above, the IA model has a set of hand-crafted weights. These weights are sometimes changed to reflect assumptions about strong, weak or false associations. But the point of interest is in exactly how the distribution of weights is formed to conform to these assumptions.

2. The problem level in the IA model clearly plays an important role, accounting for many of the interesting results. However, no account is given of why or how this particular representation should develop.

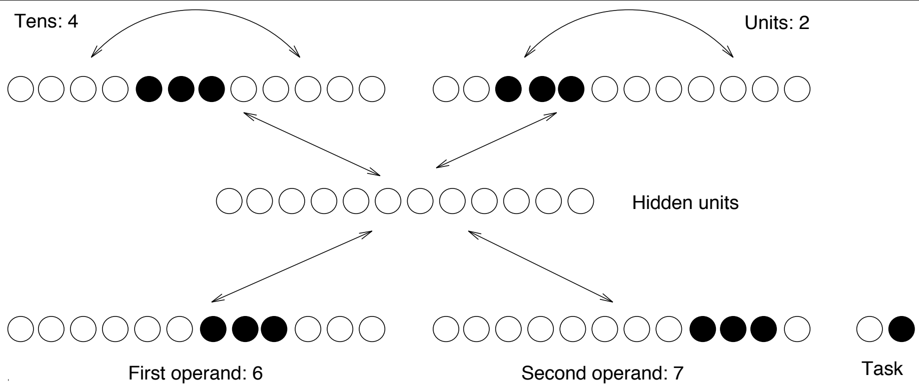

Figure 2.8. Architecture of MATHNET (from McCloskey & Lindermann 1992, figure 3).

3. The assumptions about the distribution of associations are based on the work of Siegler (1988). However, as described in section 2.2.1, there are problems with this account.

4. No detailed RT results are given. One of the methods used to account for the problem-size effect was to decrementing the weights of the “large” set of problems. Does this mean that only two different RTs could be generated, one for the small problems and one for the large? The other modification of adding false associations was done independently of the decrementing of weights, and it is not clear how these two methods interact without a detailed comparison to human human RT data.

5. No account is given for the speed of the tie and 5s problems.

6. No account is given of zero- and one-times problems.

7. The IA model is particularly let down by the lack of an explicit mechanism to account for incorrect retrieval. Without a mechanism for this the system cannot actually produce errors, regardless of internal activity.

It remains to be seen if the system can be modified to accommodate the above.

2.3.4 *Mean field theory*

The most well-informed of all the connectionist models is MATHNET (McCloskey & Lindermann 1992). The architecture is shown in figure 2.8, and is based around mean field theory (Hertz, Krogh & Palmer 1991, chapter 2) which is related to Boltzmann machines (Hinton & Sejnowski 1986).

The 26 problem nodes are divided in to two groups of 12 nodes to encode the two operands, and 2 nodes to encode the task—although only multiplication problems are currently considered. The inputs feed to 40 hidden nodes. The output layer consists of a set of 12 nodes to encode the tens part of the answer, and another 12 to encode the units field. The output nodes are connected to each other. All weights in the system are bidirectional.

Each operand is encoded as a bar of activity across three units. For the operands 7, 8 and 9 the input pattern would be…

```
7:  - - - - - - - + + + - -
8:  - - - - - - - - + + + -
9:  - - - - - - - - - + + +
```

…where **+** signifies an activation of 1.0 and **-** signifies -1.0. There are enough input units to encode the operands 0 to 9, but only the problems 2×2 to 9×9 were discussed. The encoding for the output fields is the same as the inputs.

Recall

In order to capture occasional errors, McCloskey & Lindermann begin with a system that is inherently stochastic. Retrieval begins when a problem is clamped on the input layer. The activation of the other units is computed asynchronously as follows. First the net input to a unit is computed:

$$\text{net}_i = \sum_j o_j w_{ij} + \text{bias}_i,$$

where w_{ij} is the weight between units i and j, bias_i is unit i's bias, and o_j is the output of unit j, calculated according to:

$$o_j = \tanh(\text{net}_j / T).$$

T is "temperature" of the system (discussed below).

Units are updated until the system stabilizes. This is defined as when all the units are within 0.1 of the maximum or minimum activation values. During processing, the temperature of the system, T, is lowered. At the start of processing T is large, which makes the tanh function behave linearly. As T decreases, the function becomes a sigmoid, and when T is smaller still, the function behaves as a threshold. At the start of processing, with a high T, units can take on any values between $+1$ and -1. By reducing T, the system is forced to make a decision—units become either on or off.

A 16 step annealing schedule was used: the temperature parameter was changed 16 times. This gives 16 pairs of parameters, stating when the change is made, and the new temperature value.

After the system has settled, the activations of the tens and units fields were individually compared to the representations for each of the digits, 0 to 9. The closest digit, measured by sum of squared error, was declared to be the output of the field.

Learning

The weights of the system are changed by computing the difference between the system running with just the inputs clamped (free running), and when the outputs are also clamped. First, the network is allowed to settle during the free running phase. For all connected units, $a_i a_j{}^{\text{free}}$, the product of activation values, is computed. Then, the system is rerun with the output units clamped to the desired activation values, and after settling, $a_i a_j{}^{\text{clamped}}$ is computed.

The weights are then changed:

$$\Delta w_{ij} = \alpha(a_i a_j{}^{\text{clamped}} - a_i a_j{}^{\text{free}}),$$

Operand errors	78.71
Close operand errors	90.98
Table errors	5.16
Operation error	0.0
Non-table errors	16.13

Table 2.3. Mean percentage error rates from three networks reported by McCloskey & Lindermann (1992). For close operand errors, the non-shared operand was within ± 1 of the correct digit.

where α is the learning rate, set at 0.003. The learning rule has the result that the free running system behaves like the clamped system, producing the desired output activations.

Initial experiments combined problem order and problem frequency. The 2s problems were trained, and then the 2s and 3s problems, then 2s, 3s, and 4s, and so on. Each problem set was trained for 5 epochs. Every time a new set of problems was introduced, the new problems appeared twice in the training set, and the previously trained problems occurred once.

After this, the second training phase manipulated the frequency of the problems. All 64 problems (2×2 to 9×9) were presented, but some problems were repeated to produce a total of 256 training patterns. Smaller problems occurred more frequently than larger problems in the training set. To implement this, the 64 problems were divided up into 7 groups based on the sum of the operands. For example, problems with a sum between 4 and 6 occurred 7 times in the training set; problems with a sum less than 9 occurred 6 times in the training set. The categorization seems arbitrary, but the overall result is a linear skew in favour of smaller problems.

Three networks (different initial weights) were training in this way. As the system is non-deterministic, each problem was presented 10 times to ensure that the facts really were learned. Two of the networks were correct on all problems, and the other network was incorrect just once.

Reaction time and errors

To simulate speeded testing conditions, the first five steps of the annealing schedule were skipped. Each of the three networks were tested on the 64 problems 30 times. Under these conditions the networks erred on 2.7 per cent of the trials. "Thus, the networks, like human subjects, showed reduced accuracy under speed pressure" (McCloskey & Lindermann 1992, p. 26). The RT of the networks exhibited the problem-size effect, but without the dip for the 5s or an advantage for tie problems.

Under these speeded conditions the networks make errors by settling into a state that does not correspond to the correct output. Error rates are reported in table 2.3. These errors compare well to those reported for adults (table 2.2). However, the proportion of table errors and non-table errors is reversed: the network shows 5 per cent table errors, 16 per cent non-table errors; for adults it is 13 per cent table errors, 7 per cent non-table errors.

No analysis of the network (e.g., the hidden layer) has been performed, but McCloskey & Lindermann suggest the following reasons for why certain errors are seen:

1. Similar inputs tend to produce similar outputs. Operand errors are prominent because the representations for problems that share an operand are more similar than problems that do not share operands.

2. Likewise, close operand errors occur because similar quantities have similar representations: the encoding for 9 is more similar to 8 than to 3.

3. Non-table errors often consist of the correct tens digit with the units digit from a related problem. McCloskey & Lindermann comment that it would be interesting to see if this is true of human non-table errors.

A number of experiments were run to determine the importance of the order and frequency of problems. Holding frequency constant and varying the ordering of problems, as described above, did not produce a problem-size effect. Instead, just varying the frequency of problems produced the best results. Hence, it seems that the main cause of the problem-size effect is the frequency of problems, not the order in which they are presented.

Damage

The results from brain-damaged subjects (section 2.1.2) were simulated by damaging the weights of the trained networks. There are many ways to damage a network. Examples include: perturbing the weights, removing units, changing the activation function or response mechanism. McCloskey & Lindermann chose to reduce the magnitudes of the weights by a random percentage, with a mean of 40 per cent. These damaged networks were then tested on each problem 30 times, using the 16 step annealing schedule.

As expected, the accuracy of the networks fell to a mean level of 79 per cent. Like human subjects, the damage was non-uniform. That is, some problems were severely damaged, whilst others were relatively unaffected. This is an interesting observation. Although the representations are distributed over the hidden units, some units and weights are clearly more important for some problems than others.

The error rates also followed the problem-size effect, showing a higher error rate for large problems than small problems. However, it was reported that there were more exceptions to the problem-size effect for the networks than for human subjects. The actual errors made showed a similar pattern to the undamaged networks: most errors were operand errors.

Unlike human subjects, the networks made no omission errors: the system always reached a stable state, and the answer could always be determined. This is probably an artifact of the response mechanism: it may be possible to change the response system so that omission errors are produced. This could be implemented by rejecting outputs for which the best match to a digit is below some threshold.

Human subjects show similar error rates for complimentary problems, such as 4×5 and 5×4. That is, if the subject's behaviour on 5×4 will be the same on 4×5. No such correspondence was found for MATHNET. As McCloskey & Lindermann note, this may be due to fundamental differences between MATHNET's representations and humans' representations: perhaps humans use an encoding that does not preserve order, like the interactive activation model. Or perhaps the discrepancy is due to subjects using other strategies—e.g., if the solution cannot be found, swap the order of the digits and try again.

Discussion

The MATHNET project is the most comprehensive of the accounts presented. It tackles the issues of errors, including non-table errors, RTs, frequency issues, brain-damage issues, and the details of the system are explicitly stated. A number of issues arise from the study:

- The mechanism which selects a response based on the digit with the closest match to the output vector could probably be implemented in terms of an additional "clean up" network. As noted activity needs to be turned into action. At the moment, the response selection is performed externally to the network, and no details are given as to how this is implemented in connectionist technology. McCloskey & Lindermann state that for undamaged networks most of the answers are unambiguous, hence a small amount of competition between the output units could be used as a response mechanism.

- It appears that non-table errors are occurring more frequently for the networks than for humans. The causes behind this need to be explored. For example, to what degree do the connections between the output units contribute to this effect? Are these connections needed at all?

- Given that the output layer was split into a tens field and a units field, it is surprising that MATHNET did not exhibit the RT dip associated with 5s problems. One would have expected the system to exploit the fact that all 5s problems end in zero or five.

- The system is run until all the answer nodes have saturated. Perhaps it would be possible for the system to show different RTs for the units and tens fields. That is, it may be the case that the system can produce the tens part of the answer before the units part (e.g., "six sevens are…forty…umm…two"). Whether humans or network exhibit this is an unexplored issue.

2.4 *Summary*

There is a pattern to the RTs and errors made by normal and brain-damaged adults solving multiplication problems. RTs are slower for larger problems, although there are exceptions to this rule, most notably the 5s and tie problems. Despite being a well-learned set of facts, adults make occasional errors on multiplication problems. These errors tend to be the correct answer for problems that share a digit with the presented problem. It appears that problems involving zeros or ones may be solved by the application of a general rule. This suggestion is supported by studies of brain-damaged subjects who show uniform impairment and recovery on zero and one problems.

A number of models have been proposed to account for the above phenomena. In general, the models lack explicit mechanisms or justifications for assumptions, such as the links between output units, or Campbell & Graham's magnitude units.

Network models appear to be best suited in this domain, and two such models were presented to describe the approach (Siegler 1988; Campbell & Graham 1985). Other models were omitted from the survey (notably Ashcroft 1987; Stazyk et al. 1982) because they add no details or assumptions to the approach that were not present in the Campbell & Graham or Siegler models.

Connectionist models improve on network models as they offer explicit learning and activation rules. However, with the exception of MATHNET, the retrieval and error processes have been poorly specified.

All the models differ in various ways: some require explicit training of false facts, others do not; some propose or require problem units, others exclude them; product units are used by some models, whereas tens and units outputs are used in others. Some of these parameters are discussed in the next chapter.

Finally, there is a clear need for more empirical work, particularly in the development of arithmetic skills, to judge the possible construction and use of problem units, tie flags, and operand representations. Studies are also needed to gain an understanding of the arithmetic environment, and in particular to estimate how often problems occur.

CHAPTER 3

Cascade Model of Memory for Multiplication Facts

This chapter describes a connectionist model of memory for multiplication facts built using McClelland's "cascade" equations (McClelland 1979; McClelland & Rumelhart 1988). It is referred to as the "cascade model" or "cascade network" (no relation to cascade correlation, Fahlman & Lebiere 1990).

The model was originally designed with two objectives in mind. First, in contrast to the BSB model, the network should be able to capture occasional errors. That is, although the network should correctly learn all the multiplication facts, it should also produce errors when under time pressure. The second objective was to minimize the number of assumptions about connection and unit types. This objective was formulated in the context of the Campbell & Graham (1985) model, where many different knowledge sources were presented. The aim was to see how many of the assumptions could be omitted.

The above aims were only selected after it was observed that the architecture could potentially account for the phenomena. The network was first used as a "slave" network, providing arithmetic facts for a multicolumn arithmetic network—the one described in chapter 5, although the fact network was not used in the final multicolumn network. The aim was to investigate what kinds of factual knowledge would be useful to the multicolumn network, hence the same technology was used to build the fact network (a multilayer perceptron trained with backpropagation). The results from the work of Campbell & Graham (1985) suggested that the fact network could be tested for RT and

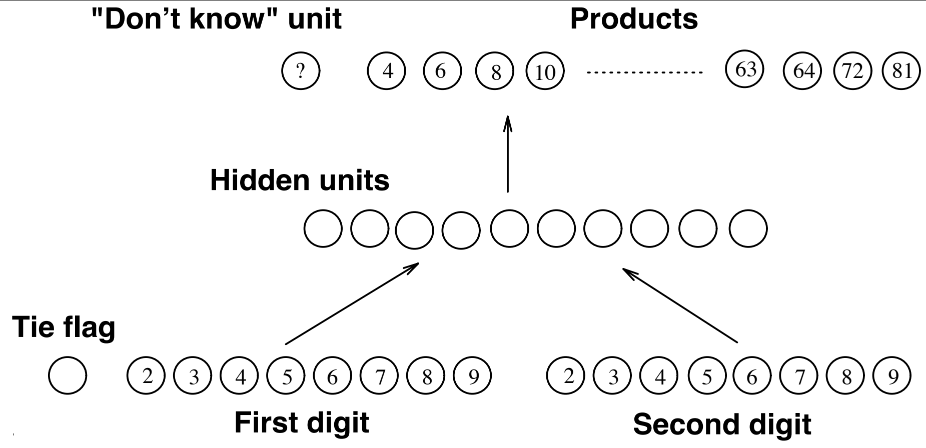

Figure 3.1. Architecture of the cascade model.

errors. When this was done, a primitive RT measure showed a dip for 5s problems, prompting further investigation of the network.

Since then the architecture and representations have changed in many ways. For example, the first experiments used a one-of-N input encoding and represented answers in separate tens and units fields. It was found that in order to capture human performance, various changes needed to be made. This chapter first outlines the "finished product", after all the changes have been made. The motivation for the changes, and the results they gave, are presented in section 3.4.

Much of the literature has focussed on the problems 2×2 to 9×9. The first set of experiments did not simulate ones or zeros problems as human empirical data had only been found for 2×2 to 9×9 at that time. However, in light of the work of Harley (1991) and Miller et al. (1984), simulations of zero and ones problems were performed. These experiments are presented in section 3.3. Finally, the system is compared to the other connectionist models, and issues arising from the model are discussed.

3.1 Architecture of the model

The structure of the network is shown in figure 3.1. The 17 inputs to the network are split into two groups of 8 units to encode the presented digits, 2–9. An additional unit is activated when a tie problem (e.g., 3×3, 4×4) is presented. The inputs are fully connected to a set of 10 hidden units. Experimentation showed that 10 hidden units are the smallest number of units which would reliably learn the problems in the training set. That is, networks with less than 10 hidden units would not always be able to learn the associations.

There is one output unit for each of the products (a one-of-N “type” encoding), plus a “don't know” unit (DKU). The tie unit and the DKU are discussed below. If a “token” output encoding was used it would be possible to construct a model without hidden units. This possibility has not been explored.

The two digits that comprise a problem are coarse encoded on the two sets of input units. Activation falls off exponentially around the presented digit. Specifically, the input to a unit, u, for a presented digit, d, is given by:

$$i_{ud} = e^{-0.5(|u-d|/0.85)^2}$$

where $|u - d|$ is the absolute difference between u and d (i.e., the distance between the two units). The constants were arbitrarily chosen so that the digit being encoded received an input of 1.0, and the immediate neighbours were 0.5. Activation continued to decay beyond the neighbours, and the input pattern is illustrated here for the digits 5, 6 and 7:

```
5: 0.00  0.06  0.50  1.00  0.50  0.06  0.00  0.00
6: 0.00  0.00  0.06  0.50  1.00  0.50  0.06  0.00
7: 0.00  0.00  0.00  0.06  0.50  1.00  0.50  0.06
```

This choice of representation is discussed in section 3.4.

For adults, tie problems are faster than their position in the multiplication table would suggest. Although Siegler (1988) found a frequency advantage for tie problems in school textbooks, simulations suggest (section 3.3.1) that this is not enough to account for the tie problems' RT advantage. So for ties an additional input unit (tie flag) is set to 1.0. Without this, the tie problems were consistently among the slowest problems for the networks. Hence, the flag is an ad hoc inclusion which exists only to allow the network to produce faster RTs for tie problems. Tie problems are difficult to account for without some change to the input encoding such as the inclusion of a tie flag. The information that the two presented digits are equal could be computed by the network: the problem is the inverse of XOR. The inclusion of a tie flag is making the information explicit. The observation here is that the tie flag speeds response on tie problems. The flag might be thought of as reflecting the perceptual distinctiveness of tie problems, possibly as a result of children learning notions like "same" and "different".

3.1.1 Recall

Networks of the kind described here usually have no reaction time: the outputs are computed in one step. A RT measure is implemented in this system by changing the activation equations.

Once the input units have been set, the "cascade" activation equation (McClelland & Rumelhart 1988, p. 153) is used to simulate the spread of activation in the network. The net input to a unit, i, at time t, is adjusted to allow activation to build up:

$$\mathrm{net}_i(t) = k \sum_j w_{ij} a_j(t) + \mathrm{bias}_i + (1-k)\ \mathrm{net}_i(t-1),$$

where w_{ij} is the weight between unit i and j. The cascade rate, k, determines the rate with which activation builds up. It is set to 0.05 in these simulations. The activity of a unit is computed with the usual logistic squashing function:

$$a_i(t) = \frac{1}{1 + e^{-\mathrm{net}_i(t)}}$$

The cascade equations can be thought of as being implemented by a unit with self-feedback. Figure 3.2 shows how activation builds up with the cascade equations. The net input to a unit determines how fast activation builds up. In figure 3.2 the system is simplified and represents a single unit connected to an input which has a fixed activation of 1. Variations in the weight produces different activation curves. The figure shows the number of iterations required to reach a fixed threshold of 0.7. With a large weight of 1.5, 16 iterations are required. With a smaller weight of 1.0, 37 cycles are needed. The activation of the negative weight decays away. This example used $k = 0.05$: a smaller value of k would result in more processing steps before the threshold is reached; a larger value of k would make it more difficult to discern events that happen in quick succession.

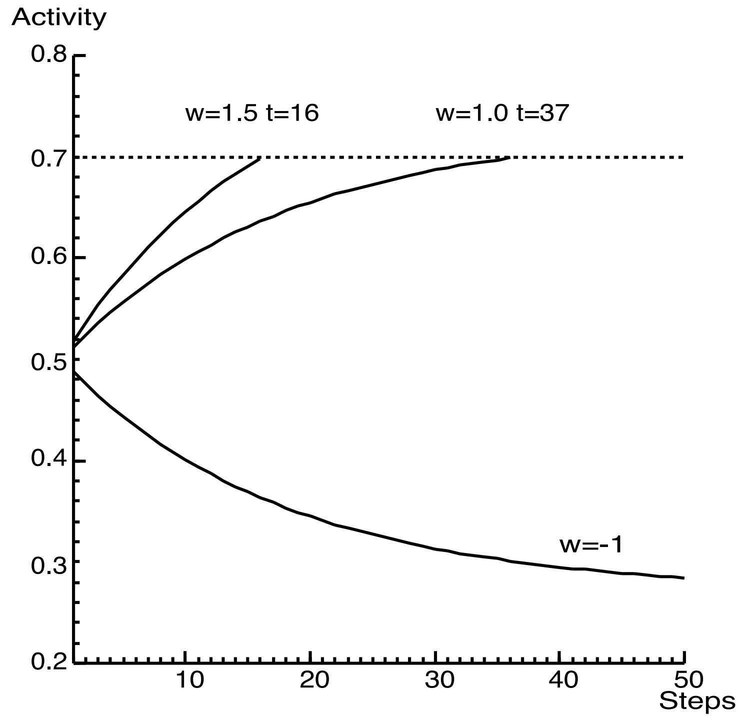

Figure 3.2. Demonstration of the cascade equations.

The RT measure is taken to be the number of steps required to reach a threshold. With a low enough threshold, certain incorrect answers will reach the threshold. This is not obvious from figure 3.2, but is demonstrated below. Errors tend to be most active early in processing, although the activation values are often small—typically less than 0.1. To allow these errors to be accepted, the threshold would have to drop from a relatively large value, say 0.7, to a small value such as 0.1. Rather than do this, the output values are normalized. That is, the response of an output unit is the normalized activation value:

$$o_i = \frac{a_i}{\sum_j a_j}$$

Here i and j refer to all the output units. There are a number of ways this normalization could be implemented in connectionist terms, but at the moment it is done externally. Throughout the chapter, and in the figures, this normalized value is used unless otherwise stated. To summarize: first the net input is computed; this is fed into the logistic function to give the activation; the activation is normalized to give an output signal. Processing continues until the activation of a product output unit exceeds a specified threshold.

To avoid any initial bias towards particular outputs, the activity of the network is started from a neutral state. Following McClelland & Rumelhart (1988), the initial state of the network is the state that results from processing an all-zeros input pattern. Note that it is not usually enough to start the system by setting the hidden vector to all zeros: output units may be selective to the non-activity of certain hidden units.

Originally, the network was trained to turn off all output units when all the input units are off. However, it was found this did not quite happen: certain products were slightly active. These products had an advantage over the others, and were always more active early in processing—just the situation that was to be avoided. This artifact was removed by adding a "don't know" unit to the output layer. The network was trained to only activate the DKU for an all-zeros input. After training, the DKU had a positive bias, and all the product units had a negative bias. The DKU appears to solve the bias problem, as simulations show that individual products do not have any advantage over other products. Thus, the DKU is a computational consideration, and it is not clear what role it might play in the equivalent human system. The DKU is not to be confused with subjects responding "don't know" to problems, or omission errors in general.

3.1.2 Training

Two sets of experiments were run. In the first, the system was trained on all the problems 2×2 to 9×9. The second experiment expanded the architecture to cover 0×0 to 9×9. The experiments are described separately below. In all cases the problems were trained in a random order using backpropagation (learning rate 0.01, momentum 0.9).

During training, the presentation frequency of each pattern is skewed in favour of the smaller problems. The skew was produced by storing the relative frequency (between zero and one) of a problem alongside the problem in the training set. When a problem was presented to the network, the weight error derivative was multiplied by the relative frequency value for that pattern. This can be thought of as providing each input pattern with a different learning rate. This method allowed accurate control over the presentation frequencies, without duplicating entries in the training set.

A different skew was used for each of the experiments, but the all-zeros pattern was always trained with a maximum relative frequency of 1.0.

Although small problems do occur more frequently in textbooks, there is no reason to believe this skew continues into adulthood (McCloskey, Harley & Sokol 1991, p. 328). To see if the effects of the skew would continue once the skew was removed, trained networks were further trained on problems with equal frequencies. That is, in both experiments, two kinds of networks were produced: the "skewed" networks, which were just trained on the skewed training set; and the "equalized" networks, which resulted from further training the skewed networks on a training set in which all problems occurred with equal frequency.

In both experiments, 20 different networks (different initial random weights) were trained in this way. All results presented below are mean results taken across the 20 networks.

McClelland & Rumelhart (1988) note that the asymptotic activation of units under the cascade equation is the same as that reached after a standard feed-forward pass. Hence, the network is trained without the cascade equation (with $k = 1$), and then the equation is switched on to monitor the network's behaviour during recall.

A final detail of the training is that the derivative of the activation function was changed slightly. Following Fahlman (1988), a small constant of 0.1 (the "sigmoid-prime offset") was added to the derivative. Backpropagation takes the derivative of a unit's activation into consideration when computing the weights changes. The derivative of the logistic activation function is:

$$f'(a_i) = a_i(1 - a_i)$$

which peaks when the activation of a unit, a_i, is 0.5, and tends towards zero when a_i tends towards 1 or 0. Intuitively this makes sense: the largest changes should be made to those units that are "undecided", with activations around 0.5. However, for some problems the activations may saturate prematurely. A small derivative slows learning, and Fahlman found that a network could fail to learn a training set because units had saturated early on. Adding a constant to the derivative means that the derivative, and hence the weight changes, do not get too small when error remains to be corrected. Sigmoid-prime offset was found to be essential in training networks on the multiplication tables, presumably because of the skew in presentation frequency.

3.2 *Simulations for 2×2 to 9×9*

The following experiment was performed to see how well the model could account for: the RT problem-size effect; exceptions to RT pattern; and the distribution of operand, close-operand, and table errors.

3.2.1 Training

The 64 problems, 2×2 to 9×9, were each assigned a frequency according to the following arbitrary function:

$$\text{frequency} = \frac{90 - \text{product}}{88}$$

This produces a linear skew, based on product, in favour of the smaller problems. For example, 2×2 was assigned frequency of 0.96, whereas the frequency was approximately 0.1 for 9×9. The parameters of this skew were the only ones tested, and the frequencies are similar to the ones used by McCloskey & Lindermann (1992). It would also be possible to explore the effect of basing the skew on a function other than the product of the two operands. Examples might include the minimum operand, maximum operand, or sum of operands. These variations have not been explored here.

Training on the skewed problems continued to an error criterion (total sum squared, TSS) of 0.05, taking approximately 8 000 epochs. After this, the networks were trained for a further 20 000 epochs with equal frequencies reaching a mean TSS of 0.005. At the end of each of the training phases, both the "skewed" networks and "equalized" networks correctly solve all problems.

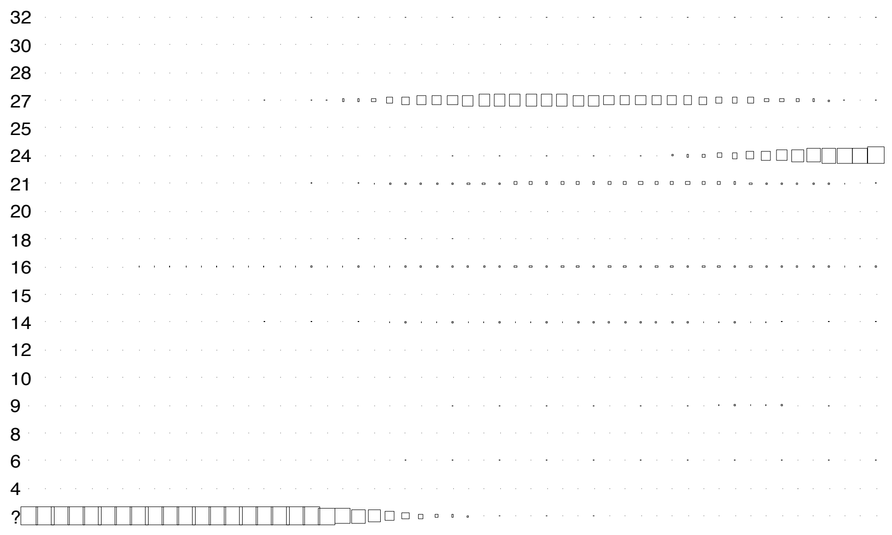

Figure 3.3. Response of the output units over 40 time steps for the problem 3×8. Output units representing products over 32 are not shown on this graph. The size of each of the squares is proportional to the output of a particular product unit at a particular moment in processing.

3.2.2 Recall

On each trial (presentation of a problem) a response threshold was selected at random from a uniform distribution in the range 0.4 and 0.9. Processing then starts from the all-zeros ("don't know") state, and proceeds until a product unit exceeds the threshold. The RT is recorded for a correct response, and erroneous responses are classified into the categories listed in chapter 2 (e.g., close-operand error, etc).

As the threshold in the recall process is a random element, each problem must be presented a number of times to capture the mean behaviour of the networks. Each network is presented with each of the 64 problems 50 times.

Given enough time (usually 50 cascade steps), the networks will produce the correct response for all 64 problems. With a high threshold, the network is allowed this time, and the correct answer is produced. However, early in processing erroneous products are active, and with a low threshold these errors are reported. Presumably the mild time pressure of the experimental situation results in subjects "lowering their thresholds".

For example, figure 3.3 shows the response of a network to the problem 3×8. After the DKU has decayed, the unit representing 27 becomes active until the network settles into the correct state representing 24. This is a demonstration of the operand distance effect, but there is slight activation of other products: 3×7=21, 2×8=16, 4×8=32, 3×3=9, and 2×7=14.

Note that the model predicts that false answers are always recalled more quickly than correct answers. This is because the false error always becomes active before the correct one. Therefore, if it is to be retrieved, the RT for the incorrect answer will be less than the RT for the correct answer. Campbell & Graham (1985) only recorded RTs for correct answers, hence it is not clear that this situation is always the case. However, it should be easy to collect evidence to refute or support the model's prediction. A finding that erroneous RTs are larger than correct RTs is strictly incompatible with the model as it stands. Of course other response mechanisms could be invented.

3.2.3 Results

The mean RTs of the networks are plotted in figure 3.4. The RTs show some of the basic features of the problem-size effect, including a dip for 5s problems.

For the skewed networks the RT correlates $r = 0.36$ ($p = 0.0018$) with adult RT (Campbell & Graham 1985). This falls to $r = 0.19$ ($p = 0.063$) after substantial training on the equalized patterns. Note that the RTs have reduced and flattened out for the equalized network, which is just what is expected after continued practice (Campbell & Graham 1985, p. 349). The obvious feature of the RT plot is the drop in RT for the nine times table. Children in grades 3 to 5 respond faster to 9s problems than 8s problems (Campbell & Graham 1985), but this levels out for adults. The 6s problems are also faster than expected when compared to human RT. Overall, the skewed network exhibit the best problem-size effect, with only a slight effect observed on the equalized networks.

The inclusion of a ties unit is necessary to ensure that ties are among the fastest problems. For the skew networks, the RTs of 6 out of the 8 tie problems were below the mean RT for their table, increasing to 7 ties for the equalized networks. 6×6 remained above the mean for the six times table.

Table 3.1 shows the error distribution for the 20 networks. This is similar to the

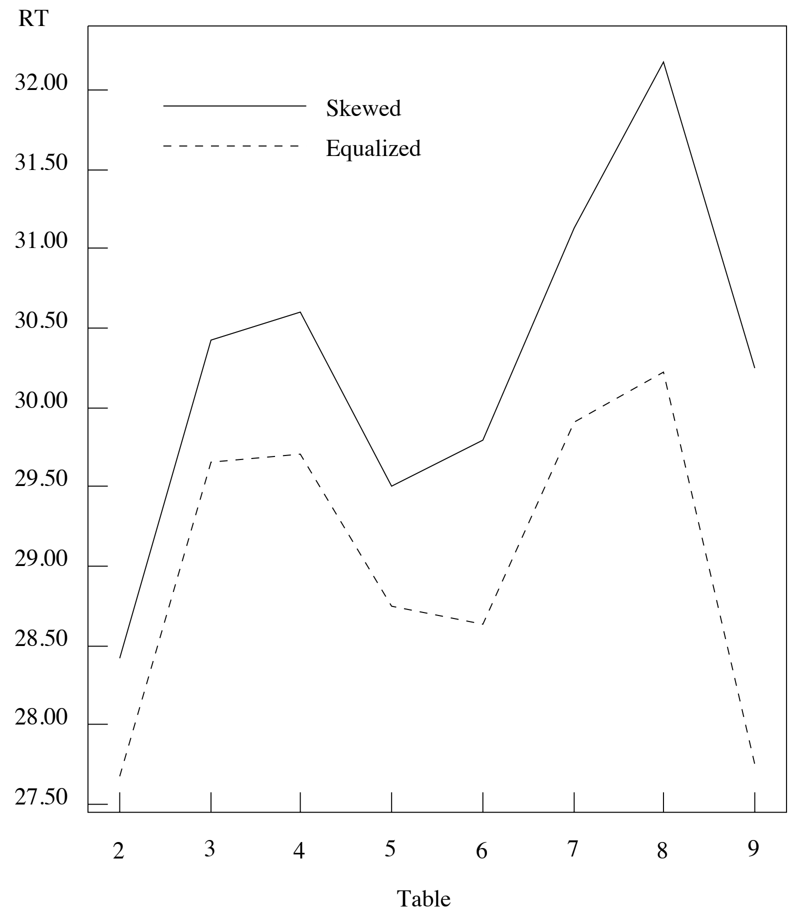

Figure 3.4. Mean correct RT per multiplication table collapsed over operand order for mean RT of 20 skewed and 20 equalized networks.

distribution for adults (table 2.1 on page 11), although without such a diversity of errors.

Table 3.2 summarizes the error distribution, and compares them to Campbell & Graham's results. Both sets of networks have error distributions that are similar to that of adults, but with a larger error rate. There is little difference between the skewed and equalized networks.

A further point of interest is the correlation between problem error rate and correct RT. Campbell (1987, p. 110) reports a correlation of 0.93 for adults. For the skewed and equalized networks $r = 0.74$ and $r = 0.76$ respectively. It is not obvious that any model would necessarily predict that slower problems produce more errors.

The model cannot produce non-table errors as all the output units correspond to products. Campbell & Graham (1985) report that only 7.4 per cent of errors are of this

	4	6	8	9	10	12	14	15	16	18	20	21	24	25	27	28	30	32	35	36	40	42	45	48	49	54	56	63	64	72	81
2×2	c	25			2					17																					
2×3	4	c	17							31																					
3×2	2	c	16							31																					
2×4		25	c		25				3																						
4×2		25	c		25				6																						
2×5					c																										
5×2					c																										
2×6	7					c				25																					
6×2	14					c				25																					
2×7							c		3																						
7×2							c		5																						
2×8		18					7		c																9						
8×2		21					11		c																17						
2×9					16					c					7															2	
9×2					14					c					13															1	
3×3		38	17	c					13	19																					24
3×4			87			c																									
4×3		1	93			c																									
3×5			29		142	11		c		27																					
5×3			35		134	7		c		20																					
3×6						51	2	9		c																					
6×3						49	1	5		c																					
3×7						5	77		45	40		c	25									9									
7×3						11	77		45	35		c	25									3									
3×8									75			24	c		64			14													
8×3									79			24	c		62			14													
3×9			5							76					c																
9×3			4							76					c																
4×4			35	10					c											34											
4×5					5			11			c												44								
5×4					2			7			c												40								
4×6						1		18		22			c				147		8	5											
6×4						1		17		21			c				144		15	3											
4×7									10			16	53			c		37	25		20	2		35							
7×4									11			16	54			c		39	25		18	2		23							
4×8									16				43					c		18	75			8					25		
8×4									16				44					c		14	78			8					25		
4×9													25		25					c			95							17	47
9×4													25		25					c			98							18	49
5×5											3			c			1						24								
5×6																	c						1			22					
6×5																	c						2			22					
5×7													10				50		c			64		41		25					
7×5													11				50		c			54		38		25					
5×8									4												c		44	20							
8×5									3												c		47	20							
5×9																	1				7		c			9					
9×5																	5				10		c			5					
6×6														3						c					24	41					
6×7																	14		13			c		32		4					
7×6																	17		14			c		30		8					
6×8																			15		7			c		22		35			
8×6																			12		8			c		25		36			
6×9																							25			c					
9×6																							25			c					
7×7							3															15			c			44			
7×8																					1			66	5		c	120	25		
8×7																								63	4		c	117	25		
7×9																								15		21		c		46	
9×7																								16		23		c		35	
8×8									6											6	22				6				c	39	61
8×9																				4						25		50	4	c	
9×8																				4						25		50	4	c	
9×9																				2								26			c

Table 3.1. Errors made by 20 "skewed" networks trained on the problems 2×2 to 9×9. "c" means correct answer, and on average represents 407 correct recalls from 500 trials.

kind, so it may not be unreasonable, at first, to focus on the other errors which make up the majority of the phenomena. However, non-table errors must be accounted for, and this topic is taken up in section 3.5.

	Networks		Adults
	Skewed	Equalized	
Operand errors	90.04	86.51	79.1
Close operand errors	78.98	73.75	76.8∗
Frequent product errors	25.0	20.49	24.2
Table errors	9.74	13.49	13.5
Operation error	3.98	3.22	1.7∗
Error frequency	14.1	18.64	7.65

∗ Approximate percentage.

Table 3.2. Percentage breakdown of errors. Figures are mean values from twenty different networks, and mean values from sixty adult subjects (Campbell & Graham 1985, appendix. A). Note that the model has not been trained on addition facts, so the frequency of operation errors is coincidental.

3.2.4 *Comments*

The following questions are addressed by analysing the trained networks:

1. Why are certain problems answered more quickly than others?

2. Why are operand errors the most frequent kinds of errors?

3. How are these behaviours established?

RT depends on the net input to a unit, and this can be increased by having some large, or many small, weights. The interesting question is why these weights develop. Presentation frequency, product frequency, initial weight values, and input encoding are all involved in determining the weights.

The presentation frequency of a problem and product should have a strong effect on the weights: those problems seen more often should develop larger weights. Simulations with networks trained on patterns with equal presentation frequencies alone (section 3.4.2) have demonstrated that the frequency of presentation is important.

However, frequency does not explain why the five times table should be faster than the four times table. "Product uniqueness" may explain why: none of the products in the five times table occur outside the context of five (unlike the two times table, where the products 12, 16 and 18 occur in other tables). Hence, the error signals for the 5s products are not diluted through differing hidden representation for different problems.

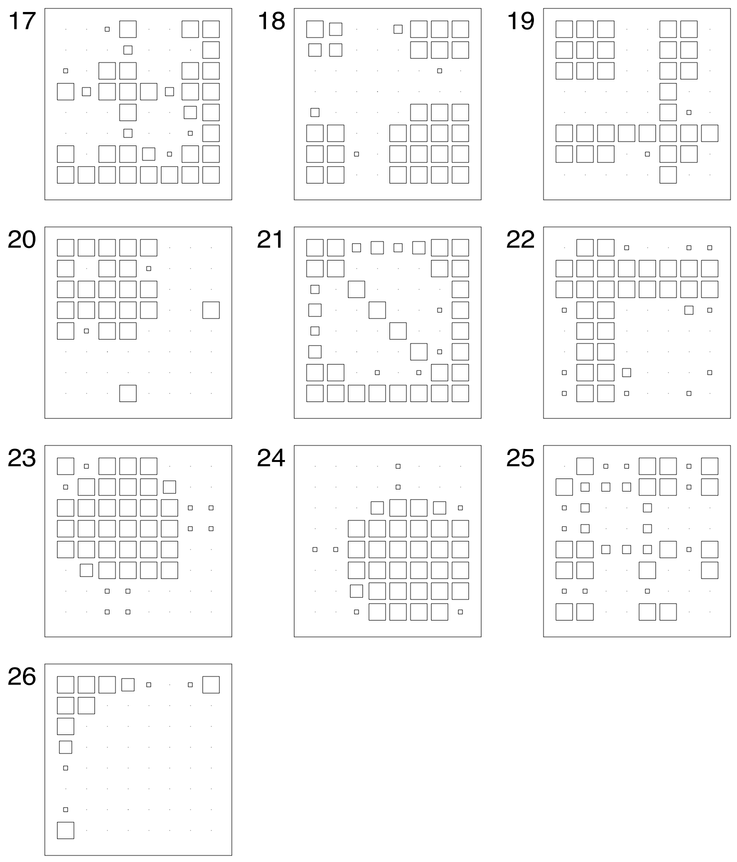

Figure 3.5. Hidden unit activations for one network. Each large rectangle represents one hidden unit. Within each rectangle, the size of the smaller rectangles represents the activation of the hidden unit to a particular problem. Each large square mimics the multiplication table (top-left for 2×2, and bottom-right for 9×9).

This explanation is not satisfactory because the same argument should apply to the seven times table as 7s products are also "unique". The 7s problems have a lower presentation frequency which may explain the discrepancy. Also, other simulations which change the product distribution (e.g., by introducing 10s problems) offer some support for the explanation. These simulations are described in section 3.4.

In this model the hidden unit encodings are learned (unlike the problem units in the interactive activation model, or magnitude units in Campbell & Graham's model). The "receptive fields" of the hidden units for one network are plotted in figure 3.5. Although there are no dedicated problem units, certain units do respond to sets of problems (e.g., unit 22 for 3s and 4s problems). One unit, 21 in this example, responds to tie problems. Other units are rather like Campbell & Graham's general magnitude units: unit 26 responds to small products; unit 23 to medium products; and unit 24 responds to larger products. The weights between the input layer and the hidden layer are approximately the same for the two operands. This is reflected in the symmetry of the receptive fields.

The hidden units tend to respond to bands of inputs. This seems to be due to the coarse coding scheme used for the inputs. Section 3.4.1 explores this point by looking at the receptive fields that result from a one-of-N input encoding. The broad response of the hidden units could be one of the causes behind the operand distance effect. Hidden units' activities change smoothly during the course of processing, but at differing rates. This affects groups of related products due to the overlap in the hidden encoding (e.g., between unit 23 and 24). The result is that some combinations of hidden unit activity may force incorrect products to exceed threshold.

Other causes for the problem-size effect and operand errors are discussed in section 3.4.

3.3 Simulations for 0×0 to 9×9

These simulations were performed for two reasons: first, to see how zero and ones problems interact with the results from the previous experiments; and, second, to use more realistic problem frequencies.

3.3.1 Training

In this experiment the presentation frequencies were skewed according to the problem frequencies reported by Siegler (1988, table 4). The frequencies are shown in figure 3.6, and are derived from the number of occurrences of each problem in second- and third-grade textbooks. These frequency values are, presumably, more "realistic" than the previous skew function that was assumed. Note, though, that there is a general increase in frequency from small to large problems. Note also the difference in frequency for zero and ones problems compared to the other problems. The contrast to other facts suggests that zeros and ones problems are treated as different kinds of problem when taught.

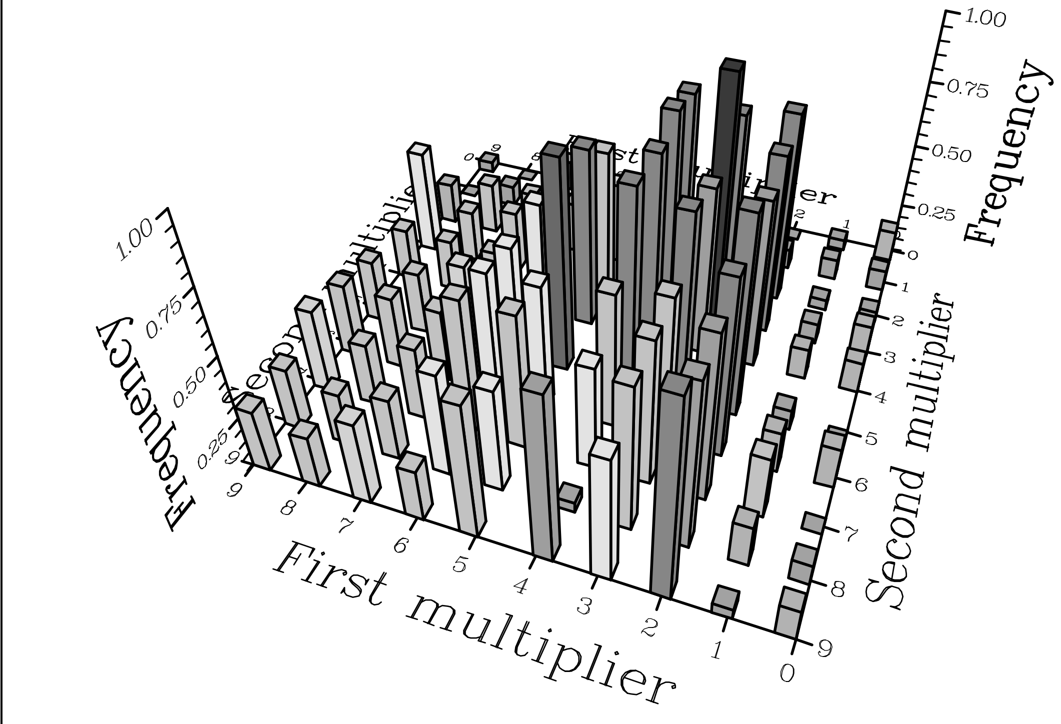

Figure 3.6. Problem frequencies (from Siegler 1988). The *x* and *z* axes represent the multiplication table (rightmost corner is 0×0, leftmost is 9×9). The *y* axis shows the frequency of each problem.

To accommodate the extra digits on the input layer, four extra input units were added—two for each operand to encode zero and one. The output layer was increased to cover the extra products that can be produced from zero and one multipliers. As the training set was increased from 65 to 101 problems (including the all-zeros pattern), it was found that an extra 2 extra hidden units were required. The network contained 21 inputs, 12 hidden units and 38 output units.

As before, skewed and equalized networks were produced. The skewed networks were trained for 35 000 epochs to a mean TSS of 0.07. A further 35 000 epochs with equal frequencies were used to produce the equalized networks, reaching a mean TSS of 0.002. Again, after training both sets of networks could correctly recall all the problems.

There was no change to the recall process.

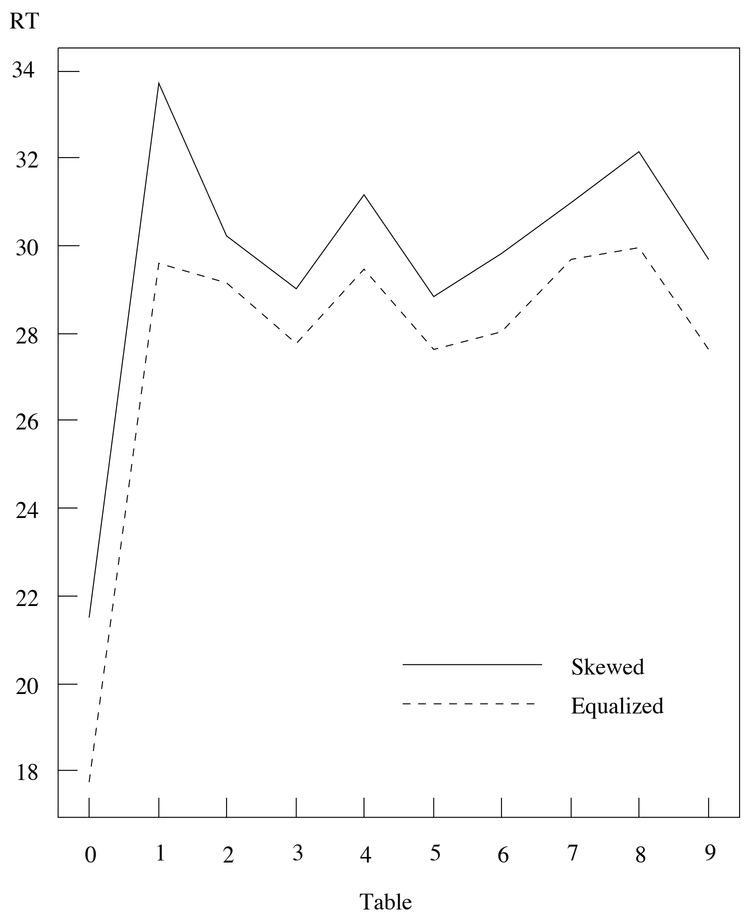

Figure 3.7. Mean correct RT per multiplication table collapsed over operand order for mean RT of 20 skewed and 20 equalized networks.

3.3.2 *Results*

For 2×2 to 9×9, the results from this experiment are similar to the previous one. The mean RTs plotted in figure 3.7 show some of the basic features of the problem-size effect, but these do not appear to be as good as the RT curves from the previous experiment. However, for the skewed networks the RT correlates $r = 0.22$ ($p = 0.013$) with adult RT reported by Miller et al. (1984). This increases to $r = 0.37$ ($p = 0.000067$) for the equalized networks. Slightly lower correlations were found with the Harley (1991) RTs.

The zero problems are solved very quickly. However, the ones problems are the among the slowest problems, which is surprising. The 3s dip is also unexpected.

Again, the RTs have reduced and flattened out for the equalized networks. Also the 5s and 9s dips are present, although the dips are comparable to the 3s dip. The dip seen

	Networks		Adults
	Skewed	Equalized	
Operand errors	93.56	93.15	86.2
Close operand errors	78.14	74.12	76.74
Frequent product errors	21.11	18.76	23.26
Table errors	6.43	6.85	13.8
Operation error	2.21	1.81	13.72
Error frequency	10.64	15.58	6.3

Table 3.3. Percentage breakdown of errors. Figures are mean values from 20 different networks, and mean values from 42 adult subjects (Harley 1991, appendix B). Adult scores other than error frequency were recomputed from Harley's data.

for 6s in the previous experiment is not as prominent here. All the tie problems were below the mean for their table, except for 6×6. The correlation between RT and error rate was $r = 0.74$ for the skewed networks and $r = 0.76$ for the equalized networks.

A comparison of error types is presented in table 3.3. The high frequency of operand errors in the "adult" column is due to a large number of errors of the form 0×N=N. The network made no errors of this form—in fact the networks made no errors at all on the zeros problems. However, the networks made a number of unrealistic errors by producing zero as an answer to many problems. This is shown in table 3.4: the first column shows zero as an answer for problems such as 4×6, 5×7 and 7×9.

3.3.3 *Comments*

This experiment has shown that the speed of the zeros problems can be accounted for within an associative network. Hence, contrary to various authors (Campbell & Graham 1985; Miller et al. 1984; Stazyk et al. 1982), it is not the RT of zero problems that suggest zeros are solved by a rule-based mechanism: it is the error pattern. Human subjects make errors of the form 0×N=N. For the network, the zero problems were very well-learned, resulting in no errors on zero problems at all. However, zero was unreasonably promoted as an error for some problems (e.g., 5×5=0).

It is conceivable that an interaction with addition would change the system's behaviour on zero problems. The introduction of problems such as 0+1=1, 0+2=2, and so on, might have two effects. First, there would no longer be a simple mapping of "anything

	0	1	2	3	4	5	6	7	8	9	10	12	14	15	16	18	20	21	24	25	27	28	30	32	35	36	40	42	45	48	49	54	56	63	64	72	81
1×1	67	c	25		8					25																											
1×2		24	c				16																														
2×1		20	c				18																														
1×3	192			c			50		14																												
3×1	193			c			50		15																												
1×4	165			7	c	7			5																												
4×1	163			6	c				1																												
1×5	58					c																															
5×1	64					c																															
1×6	105						c					14																									
6×1	103						c					11																									
1×7	41	20					25	c																													
7×1	47	21					25	c																													
1×8	143	8							c	18																											
8×1	142	4							c	19																											
1×9	96									c																								24			
9×1	94									c																								19			
2×2		33			c					22						19																					
2×3				31			c									20																					
3×2				38			c									17																					
2×4				1	36		39		c		25	43		1																							
4×2					39		40		c		25	49		8																							
2×5						15					c	3		23		49																					
5×2					2	15					c	3		23		50																					
2×6							105					c	8			25																					
6×2							115					c	8			25												2									
2×7							12	44				48	c															3									
7×2							4	49				49	c															4									
2×8		5							41	25			25		c	24																				4	
8×2		8							44	25			25		c	25																			1	8	
2×9												10				c					34																
9×2										2		3				c					41																
3×3	25			3						c											50																23
3×4	18											c				25					32	24															
4×3	18											c				22					38	20															
3×5							6				23	75		c		25			25																		
5×3							4				23	77		c		25			25																		
3×6							29					85				c		25										5									
6×3							27					75				c		25	5									3									
3×7	17											60				20		c	16			47						2						7			
7×3	15											61				15		c	18			47						2						12			
3×8																			c		76															10	16
8×3																			c		72															5	13
3×9										3						14			6		c																5
9×3										2						15			8		c																5
4×4		11								29		25		7	c					38						35											
4×5	11													25			c						24						19								
5×4	2													25			c						25						26								
4×6	11											10					25		c				9					4									
6×4	7											16					25		c				34					8									
4×7	29																	29	25			c						10			25						
7×4	30																	30	25			c						14			25						
4×8															34				2					c			21		25								
8×4															33				5					c			17		25							2	
4×9																					40					c			25			3					
9×4																					44					c			25			2					
5×5	7																			c						2			38								
5×6																							c									9					
6×5																							c									14					
5×7	25																						25		c			25									
7×5	25																						25		c			25									
5×8																											c		80								
8×5																											c		68								
5×9																													c			25					
9×5																													c			23					
6×6	23									19											5					c			17	9		52					
6×7																												c		23		18					
7×6																												c		21		16					
6×8																														c		87					
8×6																												8		c		88					
6×9																											1					c					
9×6																											1					c					
7×7	46																									25		24		25	c	5	15	7			
7×8	25																													27	10		c	90			
8×7	25																													34	14		c	89			
7×9	14																										10			9		16		c			
9×7	13																										14			10		13		c			
8×8		9							23																					9	10				c	50	31
8×9																																		23		c	
9×8																																		24		c	
9×9																													9						7	19	c

Table 3.4. Errors of 20 "skewed" networks. No errors were made on zero problems, and these have been omitted from the table. "c" means correct answer, and on average represents 423 correct recalls from 500 trials.

involving a zero is zero", reducing the prominence of zero as an unlikely error for some problems. Second, the addition problems involving a zero may increase the likelihood of a 0×N=N error. On the other hand, the inclusion of addition problems may increase the RT for zero multiplication problems. This speculation should be backed by simulations and requires more thought (e.g., how to incorporate the two operations).

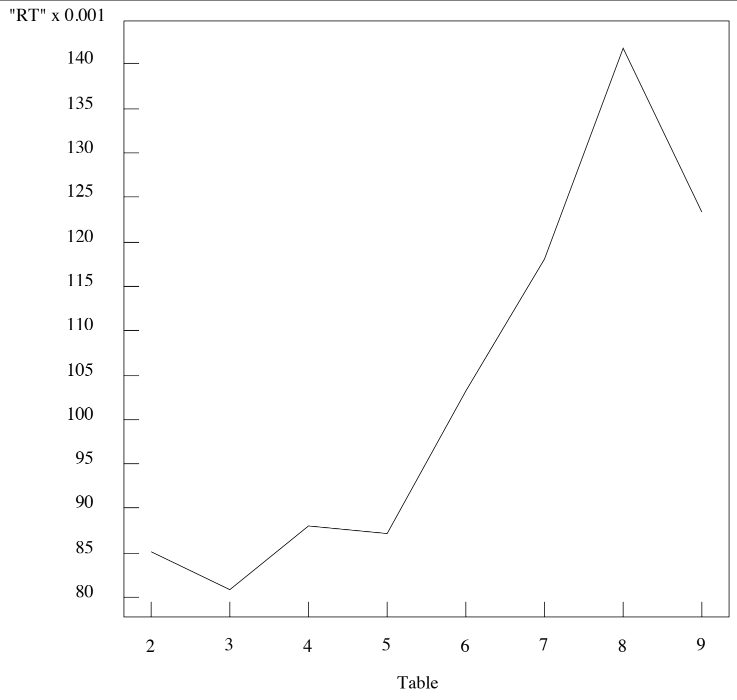

Figure 3.8. RT derived from the problem frequency shown in figure 3.6. The RT is the reciprocal of the mean frequency of problems in each table.

The slow RT of the ones problems may be due to the low frequency of the problems in the training set. This seems likely given that the RT for these problems decreases markedly after further training, as shown for the equalized networks. However, the ones RT should be comparable to the zeros problems (cf. figure 2.3 on page 16). There are numerous ad hoc ways in which the network could be made to produce faster RTs for ones problems. For example, the input representation could be biased towards zero and ones problems, perhaps by using a logarithmic input encoding, or just allocating more input bits to smaller operands. Again, more work is needed here.

The change from using a linearly skewed training set to one based on Siegler's analysis produced a dip in the RT for 3s problems. This dip can be explained by examining the presentation frequencies: there is a frequency peak for 3s problems in the training set. Assuming that RT is dependent on problem frequency (which is, of course, only part of the story), it is possible to plot the RT derived from the problem frequencies. Figure 3.8 is a plot of the reciprocal of mean problem frequency per table: the idea being that the more frequent problems require less time to respond. The graph clearly shows the 3s dip. All

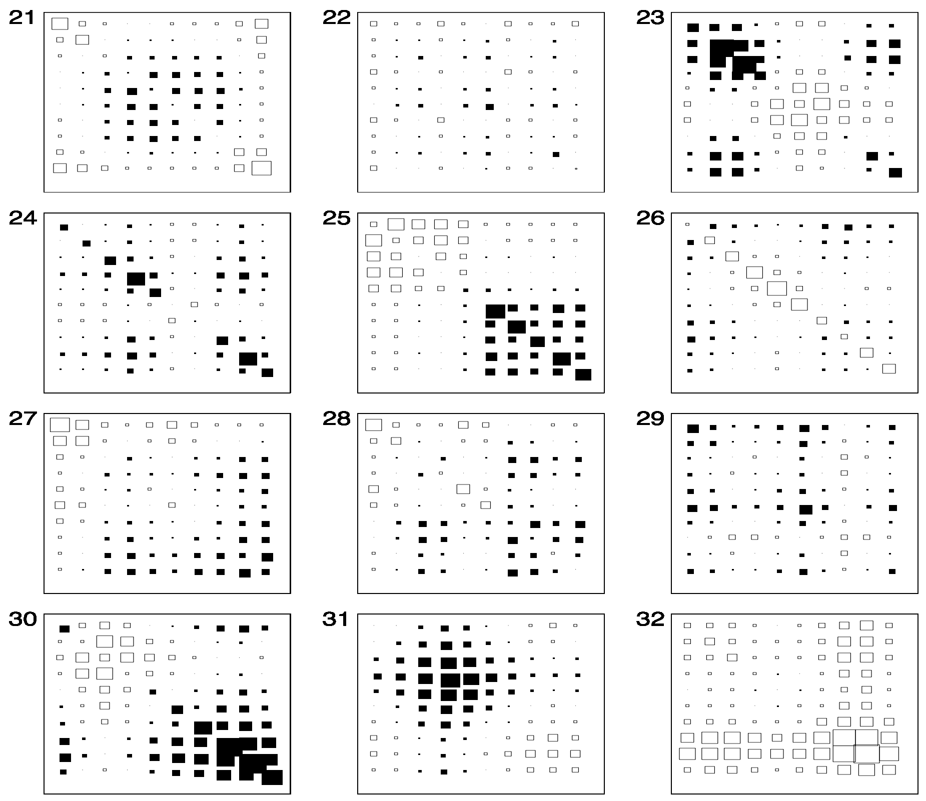

Figure 3.9. Net input to a network's hidden units. Each large rectangle represents one hidden unit. Within each rectangle, the size of the smaller rectangles represents the net input to the unit for a particular problem. Negative net input is shown by filled squares, and the size of the square indicates the magnitude of the net input.

this suggests that the networks are heavily affected by changes in problem frequencies.

The presentation frequency also peaks slightly for the five times table. This may be the reason behind the speed of the fives, but earlier simulations using a linearly skewed frequency curve also produced the characteristic 5s dip.

Similar hidden units developed in the two experiments. Figure 3.9 displays the net input to each hidden unit for a network (unlike the previous hidden unit graph, which showed activation). Unit 27 responds to problems involving a zero, but otherwise the responses are similar to those for the previous experiment.

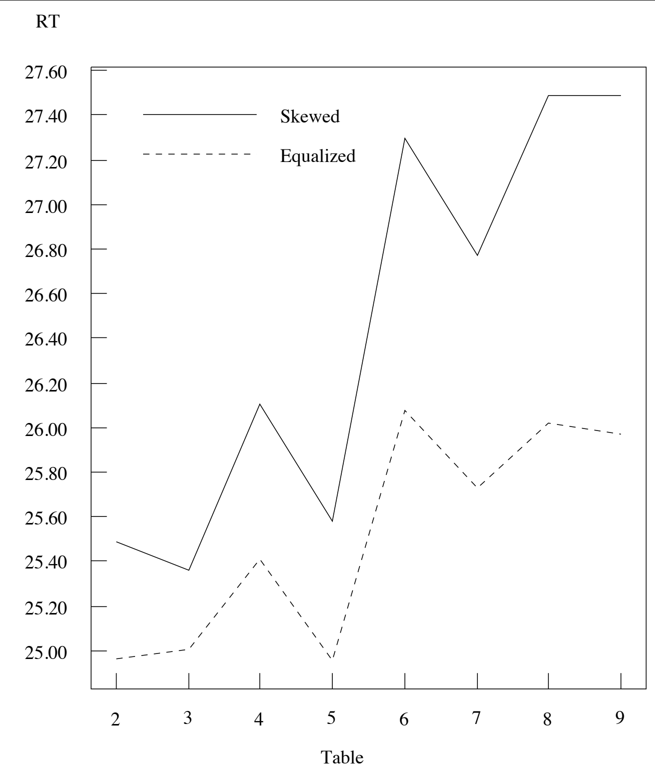

Figure 3.10. RTs from simulations of 10 networks trained with a one-of-N input encoding.

3.4 Further experiments

The simulations presented above show that the response mechanism produces errors that are comparable to human errors. RTs also tend to be faster for smaller than larger problems, but exceptions to this rule are exhibited. Some of the causes of this behaviour are discussed below in the context of variations in the experiments.

3.4.1 One-of-N input encoding

One of the original set of simulations run for the problems 2×2 to 9×9 used a one-of-N input encoding. That is, just one input unit was activated for each operand, rather than using the coarse encoding. Other than the change to input encoding, the rest of the simulation was the same as the 2×2 to 9×9 experiments reported above.

Figure 3.10 shows the mean RT from simulations with 10 different skewed networks and 10 equalized networks. The RT curves from this experiment were exceptionally good, correlating $r = 0.6$ with the adult RTs reported by Campbell & Graham (1985). Note the 7s dip that is expected from the "uniqueness" argument presented in section 3.2.4.

The errors were predominantly operand errors (around 97 per cent). However, only 58 per cent were close-operand errors, compared to 76 per cent for adults, and only 2 per cent were table errors, compared to 13 per cent for adults. These observed discrepancies lead to changing the input representation to the coarse encoding described in previous experiments.

Note here that the networks trained with the one-of-N encoding do not need to be trained on false associations to be able to produce errors. This is in contrast to the majority of the models described in chapter 2: those models reply on false associations as a source of errors. When the input encoding was changed to a coarse encoding the proportion of close operand and table errors increases.

Previously I have commented (Dallaway 1992a) that the cascade model, when using the coarse encoding, does not need to be trained on false associations to model the phenomena. Yet, the coarse coding of operands implicitly introduces false associations. For example, training on 5×8=40 does not just associate the 5 and 8 input units with the 40 output unit. Rather, half associations are also formed between 4×8=40, 6×8=40, 5×7=40 and 5×9=40. This is because the 4 and 6, and the 7 and 9 units are half-activated while the other operand is fully activated. This is, of course, just the pattern needed to account for close-operand errors (not operand-errors for 5×8, but for the other four problems). Using the coarse representation, even smaller associations will be formed for 4×7=40, 4×9=40, 6×7=40 and 6×9=40. In this case both operands are only half-activated, giving a much smaller association. This second pattern captures table errors. It seems, then, that the change in input encoding introduces false associations which are suitable for increasing the percentage of close-operand and table errors in the cascade model.

The input encoding that McCloskey & Lindermann (1992) used for MATHNET (a bar of 3 units) should make all these false associations with equal strength. Hence it is surprising that only 5 per cent of MATHNET's errors were table-errors, and more errors were non-table errors. The output encoding for MATHNET consists of tens and units fields, rather than products. The interaction between different input and output encodings may complicate matters (e.g., input patterns are not attempting to activate particular products, but are instead activating particular tens and particular units.)

Although the input encoding used by the cascade model (and MATHNET and BSB) can be conceptualized as a coarseness in the encoding of operands, it can also be seen as introducing false associations. The improvements to the model brought about by introducing a coarse coding could also be effected by explicitly training on an appropriate set of false products. As such the model blurs any theoretical difference between the learning of false associations or the coarseness of operand encoding. That is, the current model does not offer a way to choose between the two options, or of deciding how much each contributes to the phenomena. Hence in some ways it does not make sense to ask whether errors are caused by a coarseness in operand encoding *or* by false associations: both produce false associations. Deciding if children actually learn false associations from their environment requires further empirical studies. Likewise, deciding if children learn false associations as a result of having a particular kind of operand representation also requires further empirical studies.

The receptive field plots that result from a one-of-N input encoding show some interesting differences to the plots for coarse coding (figure 3.11). The encodings are "tighter"—they do not show the same degree of broad response that the coarse encoding produces. Note also that the hidden units are responding more selectively for certain tables. For example: unit 17 is responding to 6s; unit 18 to 3s; unit 21 to 9s; unit 23 to 7s.

These results indicate that the degree of coarseness (or sharpness) of the encoding is something that needs to be explored. The sharpness of the encoding appears to play a role in RT (good for sharp, one-of-N encoding), errors (better with some degree of coarseness) and hidden unit representations.

3.4.2 *Training without a frequency skew*

To asses the importance of the skew in problem frequency, 10 networks were trained with no frequency skew. The resulting mean RTs are shown in figure 3.12. Without a frequency skew the RT curve lacks the problem-size effect. As expected, then, the skew offers an advantage to smaller problems. The figure can be thought of as representing the inherent

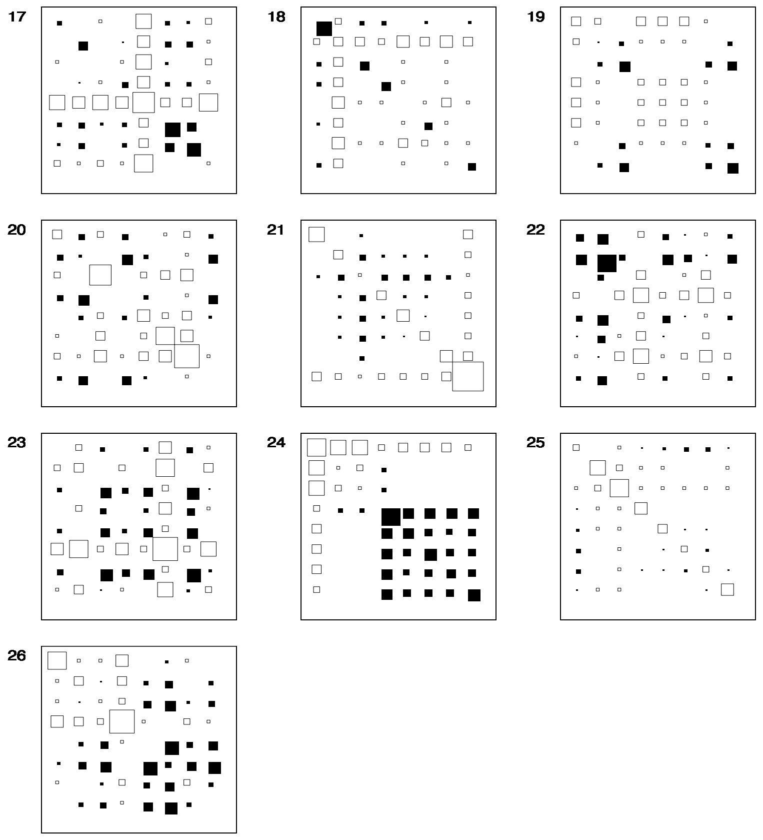

Figure 3.11. Net input to a network's hidden units with one-of-N input encoding. Each large rectangle represents one hidden unit. Within each rectangle, the size of the smaller rectangles represents the net input to the unit for a particular problem. Negative net input is shown by filled squares, and the size of the square indicates the magnitude of the net input.

difficulty of the input-output mappings in the training set: 3s, 5s and 9s appear to have a natural advantage over other problems.

The coarse input encoding was used, and the errors made by the networks were mostly operand errors. Of all trials, 26 per cent resulted in an error: 87 per cent were operand errors (81 per cent close operand); 25 per cent were high-frequency errors; 13 per cent were table errors.

3.4.3 *McCloskey & Lindermann's input encoding*

There is a problem with the coarse input encoding used in the experiments of sections 3.2 and 3.3: the encoding is biased against the smallest and largest operands, favouring the

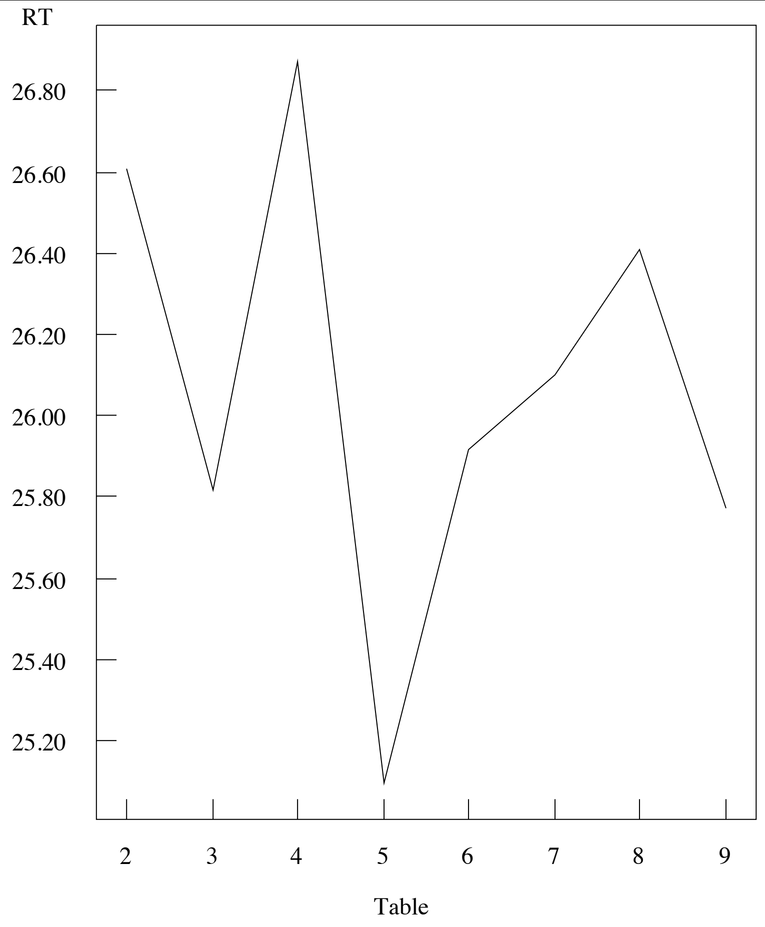

Figure 3.12. RTs from simulations of 10 networks trained with equal frequencies only.

central operands. The reason is that the input encoding decays around the operand being encoded, but for the "edge" digits (2 and 9, or 0 and 9), there are only neighbours to one side of the digit. Hence, other digits (e.g., 5 and 6) supply a larger input.

The bias towards the central digits may be one of the reasons why 6s show fast RTs. The same could be said of 5s problems, however other simulations show that the effect on 5s is not significant. The 0×0 to 9×9 experiment was re-run using the MATHNET encoding. Each operand is represented by activating 3 consecutive units across the 12 input units. The extra 2 units allow the encoding of 0 and 9 to have activation on both sides. Hence, no input pattern is favoured, except for ties which are intentionally favoured.

The results from these simulations are similar, and arguably better, than the results from the previous input encoding. Table 3.5 summarizes the errors, and figure 3.13 shows the RTs.

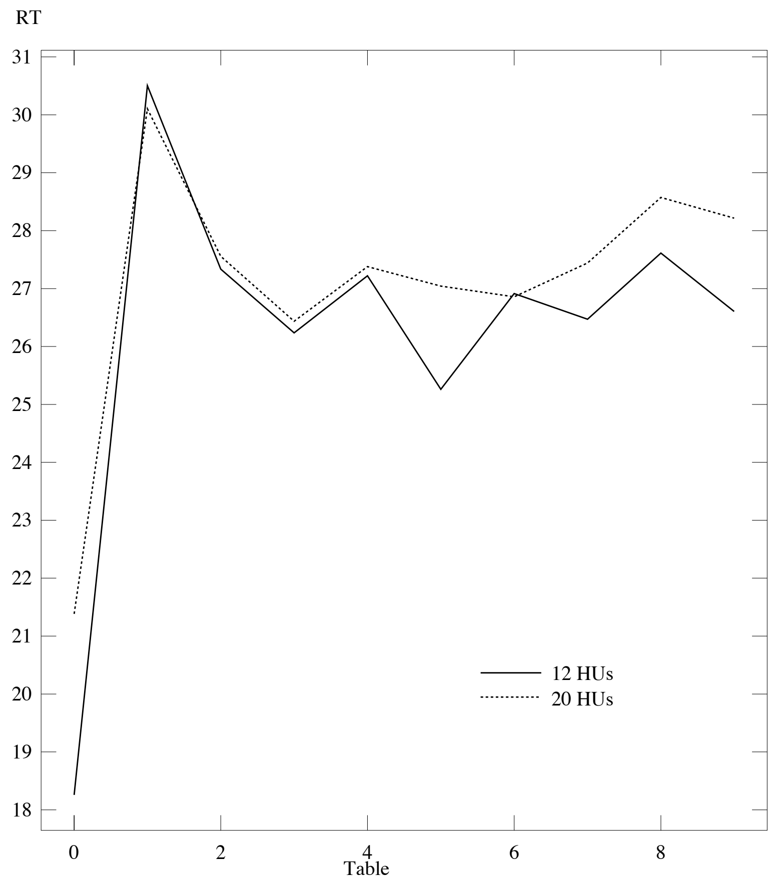

Figure 3.13. RTs from simulations of 20 skewed networks with 12 hidden units and 20 networks using 20 hidden units. Both networks used MATHNET's input representation.

For this experiment networks with 12 and 20 hidden units were trained. As can be seen from table 3.5, the 20 hidden unit network made much fewer errors, but all of the errors were operand related. The 5s dip vanished for the 20 hidden unit network, but the overall problem-size effect was slightly better than the 12 hidden unit network. The 5s dip is prominent for the 12 hidden unit network, and the 6s dip is absent. In previous experiments the 10 and 12 hidden units were used because this was the smallest number of unit that would reliably learn the training set. However, these results suggest that the quantity of hidden units is an important parameter for the model, and the effect of varying the number should be explored in more detail.

	Skewed nets		Adults
	12 hidden	20 hidden	
Operand errors	90.45	100.0	86.2
Close operand errors	82.08	98.32	76.74
Frequent product errors	18.07	26.62	23.26
Table errors	9.55	0.0	13.8
Operation error	1.05	0.0	13.72
Error frequency	9.51	0.83	6.3

Table 3.5. Percentage breakdown of errors for networks using the MATHNET encoding. Figures are mean values from 20 different skewed networks with 12 and 20 hidden units, and mean values from 42 adult subjects (Harley 1991, appendix B). Adult scores other than error frequency were recomputed from Harley's data.

3.4.4 Predictions for the 10, 11 and 12 tables

Children are taught the multiplication tables up to 10, and they were once taught them up to 12. Out of curiosity, the cascade model was trained on 10, 11 and 12 problems. The input and output layers were expanded to accommodate the extra numbers, and a total of 15 hidden units were used—a similar expansion to the one required to model 0s and 1s problems. The input encoding was the MATHNET encoding described in the previous section.

There are a number of uncertainties with this simulation. First, it is not clear what kinds of errors or RTs to expect from adults. Intuition suggests that: 10s problems will be solved very quickly; 11s problems may be solved almost as quickly because of the pattern to most of the problems (for N less than 10, $11\times N=NN$, e.g., $11\times 4=44$); 12s will probably be the slowest of problems. A low error rate might be expected for 10s, a high error rate for 12, and possibly some kind of rule-based errors could be observed for 11s problems.

Second, it is not obvious that two-digit operands are handled in the same way as one-digit problems. Rather than make any more assumptions the 10, 11 and 12 operands are represented in the model in the same way as the other operands.

Finally, what frequency skew should be used for the extra problems? In this simulation the Siegler skew was used for problems 0×0 to 9×9, but a linear skew was used for 10, 11 and 12. For example, a relative frequency of 0.22 was used for 10×10, down to 0.1 for

12×12. This was produced by the arbitrary function:

$$\text{frequency} = \frac{180 - \text{product}}{360}$$

Ten skewed and equalized networks were trained under these assumptions. The resultant error distribution was comparable to previous simulations (95 per cent operand errors, 73 per cent close-operand, 3 per cent table errors). No particular patterns were observed. The skewed network produced, on average, 1.11 per cent of omission errors. That is, on those trials no output reached the threshold within 100 cycles. The equalized networks produced no omission errors.

The RT graph (figure 3.14) is not unlike the ones reported for the simulations using the MATHNET input encoding. The 12s problems are among the slowest problems, and the 11s problems are solved relatively quickly, but the 10s are slower than expected. The graph shows dips for the primes 3,5,7 and 11, although the 5s dip is not present for the equalized networks. The disappearance of the 5s dip is perhaps due to the reduction in "uniqueness" of the 5s problems. The 10s problems interfere with 5s products, making 5s harder to learn. If 10s really are as easy to recall as intuition suggests, then the model offers a poor account of 5s and 10s recall. This is further discussed later.

A point to note here is that a change in the distribution of products has changed certain aspects of the RT graph.

3.4.5 Damaging the network

The results from experiments with brain-damaged subjects (e.g., McCloskey, Aliminosa & Sokol 1991) offer an additional source of findings to compare to the model. In this section results are presented from various kinds of "lesions" made to trained networks. As described in section 2.1.2, the main results from the literature include:

1. Uniform damage for zero problems. That is, the zero problems should show equal amounts of damage.

2. Nonuniform damage for non-zero problems.

3. Generally, larger problems are more prone to damage.

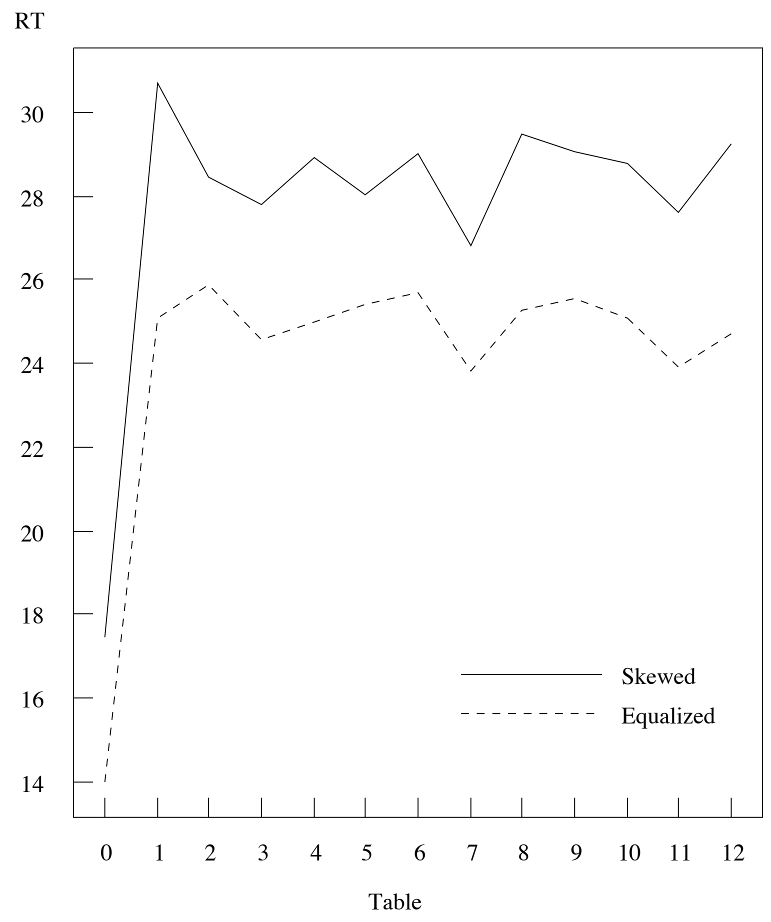

Figure 3.14. Predictions for 10, 11 and 12 times tables.

4. The distribution of errors are similar to the non-damaged subjects (e.g., errors should be mostly operand errors).

5. There is a close correspondence between complimentary problems. That is, the performance on 4×6 should be the same as performance on 6×4.

There are many ways to damage a network, including: removing units, altering weights, changing the activation functions, damaging the input or output units. The 20 skewed networks from the 0×0 to 9×9 experiments were damaged in the following ways:

1. Adding a different random value to all weights ("absolute" damage).

2. Multiplying each weight by a different random value ("relative" damage).

3. Randomly deleting a hidden unit.

	0	1	2	3	4	5	6	7	8	9
0	0	44	32	0	0	0	0	0	0	0
1	40	0	0	36	100	0	0	0	0	32
2	0	0	0	0	0	0	0	100	0	28
3	0	0	0	0	0	0	0	0	100	0
4	0	100	0	0	0	100	0	100	0	0
5	0	0	0	0	0	0	100	0	100	0
6	0	0	0	0	72	100	100	0	0	0
7	0	0	100	0	100	100	0	0	100	0
8	0	0	0	100	0	96	0	100	100	88
9	0	100	100	0	0	100	0	0	0	0

Table 3.6. Percentage error on each of the problems 0×0 to 9×9 for one network produced with "relative" damage.

4. Reducing the magnitude of the weights by a fixed amount.

Each kind of damage was tried with a variety of parameter settings. Each network was tested 25 times on each of the 100 problems, with response thresholds randomly selected between 0.8 and 0.9. This equates to simulations without any time pressure. In all cases damage resulted in an increase in error rates and in omission errors. None of the networks produced omission errors before damage.

The values of the parameters of each kind of damage were as follows: for absolute damage each weight was incremented by a random value between ±1.25; for relative damage the value was ±0.2; one hidden unit should be removed; and for magnitude reduction, each weight was reduced by 1/4.

Relative and absolute damage gave similar results. There was no correlation between error rate and product size. Omission errors occurred on an average of 37 per cent of trials. Operand errors and close-operand errors were prominent, and all errors were nonuniform. Zero problems were almost always undamaged. There was little similarity between complimentary problems. As an example, the errors resulting from relative damage for one network are shown in table 3.6. The errors include omission errors.

By deleting a single hidden unit networks could be made to exhibit non-uniform damage on non-zero problems, and showed a tendency towards uniform damage for zero problems. Seven out of 20 networks showed complete uniform damage for zero problems. That is, all zeros problems were either 100 per cent correct or 100 per cent

	0	1	2	3	4	5	6	7	8	9
0	100	100	100	100	100	100	100	100	100	100
1	100	100	0	0	0	100	0	0	0	0
2	100	0	0	0	0	0	0	0	100	0
3	100	0	0	0	0	0	0	0	0	0
4	100	0	0	0	0	0	0	0	0	0
5	100	100	0	0	0	0	0	0	0	0
6	100	0	0	0	0	0	100	0	0	0
7	100	0	0	0	0	0	0	100	100	0
8	100	0	100	0	0	0	0	100	80	0
9	100	0	0	0	0	0	0	0	0	0

Table 3.7. Percentage error on each of the problems 0×0 to 9×9 for one network produced by removing a hidden unit.

incorrect. Six networks were mostly uniform, with just 2 of the 19 zero problems showing a different degree of damage. The other networks were non-uniform on 4, 6 or 8 zero problems. Corresponding problems tended to show the same amount of damage, as can be seen from table 3.7. In general, there was no correlation between product size and error rate, and operand errors varied between 22 and 73 per cent, with a mean of just 45 per cent. Omission errors also varied greatly, with a mean of 47 per cent of all errors.

Uniformly reducing the magnitude of weights produced the following effects. There was a reliable correlation between product-size and error rate, and damage was non-uniform—including zero problems. Operand errors were variable, and omission errors were high (around 50 per cent of all errors). Complimentary problems showed similar error rates.

The results from these simulations are preliminary and inconclusive. It seems possible that deletion of just the right hidden unit can produce uniform errors for zero problems, whilst allowing non-uniform damage on other problems. Other kinds of damage show a correlation between the size of the problem and the number of errors made. Presumably a mixture of damage (e.g., some unit loss and some weight reduction) will be able to produce the behaviour seen in brain-damaged patients. However, current simulations show very high increases in omission errors and a noticeable reduction in the percentage of operand errors. The reduction in operand errors may not be serious, as McCloskey, Aliminosa & Sokol (1991) report that their patients produced operand errors on between

35 and 80 per cent of trails. The large increase in omission errors may be more important, as 5 out of 7 of McCloskey, Aliminosa & Sokol's patients showed no omission errors, and the other two omitted an answer on 46 and 24 per cent of trials (p. 177).

Further simulations are required to clarify these results. For example, it may be possible to change the parameters on the methods of damage to reduce omission errors, or just lower the response threshold. The general problem here is that there are many parameters to consider: not only parameters to the methods of damaging networks, but different ways to damage networks. There also appear to be no good reasons for choosing one kind of damage over another, or indeed, for supposing that one kind of damage simulates the damage suffered by patients.

For these experiments mean results are not generally useful. This creates a problem because one cannot easily be certain that an observed pattern of damage is typical and reliable, or just an artifact of a particular parameter setting. Rather than report mean statistics, it is necessary to classify each network in some way. It is not obvious how to form these categories, but they might include some kind of error rate uniformity measures for each table. More work is required before this aspect of the model can be systematically explored. It is nevertheless encouraging to see that non-uniform errors can be produced, and that there is some potential for producing uniform errors for zero problems.

3.5 Discussion

The preceding simulations and analysis have suggested the following causes for the problem-size effect and operand errors:

- *The input encoding.* Presented digits are associated with all the products in the digit's table, producing operand errors. Close-operand errors result from false associations formed by the coarseness of the input encoding.
- *The presentation frequency.* Smaller problems are presented more often than larger problems, and this is the foundation of the problem-size effect.
- *The nature of the facts themselves.* Certain problems, such as 5s, may be easier to learn due to the distribution of the associated products. That is, 5s products participate

only in problems involving the digit 5, unlike, say 6s products, which are found in problems where a 6 is not a presented operand. This effect may be enhanced or reduced by the choice of output encoding, or more generally speaking, by the details of the input-output mapping to be learned in the context of the recall architecture.

The simulations have shown that the model is affected by these parameters. The change from a one-of-N input encoding to a coarse encoding changed the simulation results. Changes in presentation frequency were not explicitly explored, and were compounded with changes to the problem set brought about by the introduction of zero and ones problems. The distribution of products was modified with the introduction of 10s, 11s and 12s problems. This brought about a change in the RT results. Future work should look at how systematic changes to these parameters change the performance of the model.

Some of the general issues that arise from the model are discussed below. The next section looks at the choice of output representation. Section 3.5.2 discusses the problems posed by the zero and ones problems. Section 3.5.3 speculates as to how verification and primed production tasks could be incorporated into the model.

3.5.1 Choice of output representation

Non-table errors are not accounted for because the outputs of the model are represented as product nodes. This choice has a number of consequences. First, it means that a separate read-out mechanism is required to capture non-table errors. One possible scheme would be to add a tens layer and a units layer above the product layer in the network. The details of this scheme have not been worked out, but one could imagine that the system would be similar to the current model. Each unit or ten node would receive a different number of connections from the product nodes. For example, the tens unit representing 60, would be connected to 2 products (64 and 63), whereas the 70s unit would receive input from just the 72 product. In this way, each of the tens and units nodes would show differing RTs and error rates. Whether this system produces table-errors while preserving the distribution of the other types of error remains to be seen. Note, though, that the scheme brings added complexity: is RT measured when either the tens of units fields exceed a threshold, or both? Does the read-out mechanism operate in parallel with the rest of the system, or is it used after a product unit exceeds threshold?

Results from the work of Harley (1991, pp. 99-108) support the plausibility of a separate read-out mechanism. Harley's experiments measured the table-error rates of subjects solving multiplication problems, and the error rates for just reading numbers. The analysis of the results suggested that non-table errors occurred in 0.14 per cent of reading trials. Multiplication produces 0.52 per cent non-table errors. This is not a statistically significant difference in error rates. As such there is no reason to reject the null hypothesis that non-table errors from reading and recall result from the same mechanism. Presumably this shared mechanism is not part of fact recall, and is more likely to be part of some kind of read-out mechanism—the number production element of the modular structure of number processing shown in figure 1.1. To pursue this argument further requires the specification of an explicit read-out mechanism. This aspect of the model is an avenue for future study.

A second point of interest surrounds the formation of product units. It is not unreasonable to suppose that certain, often encountered, numbers are represented as single nodes. However, there are limits to this representation, but it is not obvious what these limits are. At some stage a more general "parsing" mechanism may be employed to handle longer numbers, and it may even be related to the read-out mechanism. It is assumed here that memory for simple arithmetic facts is a modular system, but at some stage the connection between product units and other representation of number should be explored.

The cascade model does not attempt to explain the formation of product units, and simply assumes that product units are pre-formed. How and why certain numbers are represented in this way is one of the many questions not addressed here.

3.5.2 Rule based processing

Error results from normal and brain-damaged subjects indicate that zero and ones problems are not handled in the same way as other multiplication facts. To recap, RTs for zero and ones problems are low, and zero errors tend to be of the form $0\times N=N$. Brain-damaged subjects show uniform impairment on zeros problems, and recover the ability

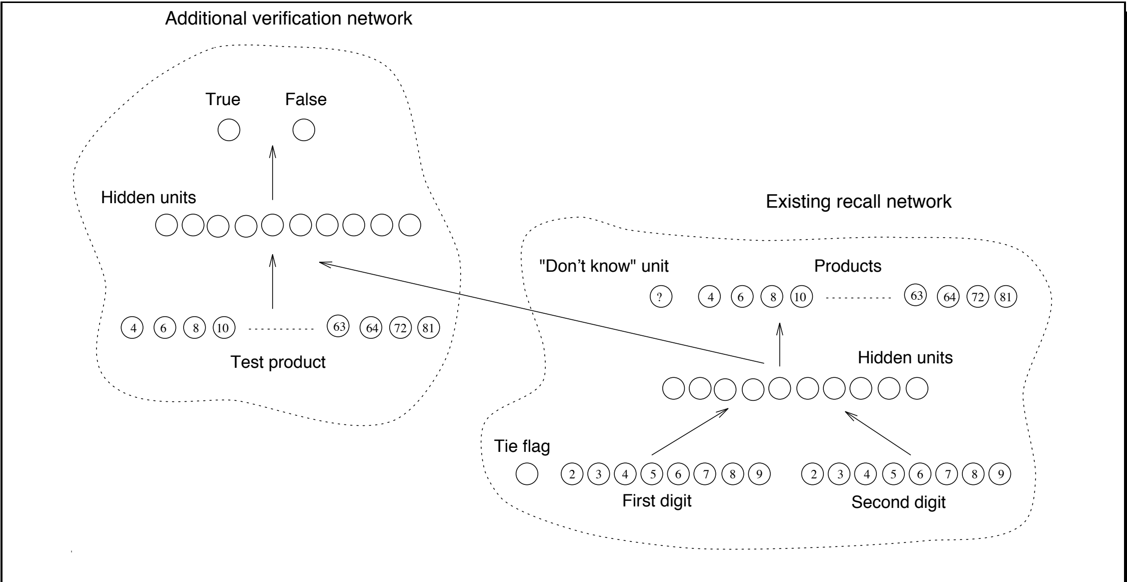

Figure 3.15. Possible account of verification for the cascade model.

to solve zero problems after remediation on just two zero problems. The cascade model, although fast on zero problems, is not fast on ones problems, and pays a penalty of offering zero as an answer to many problems.

Section 3.3.3 speculated as to how the model could be modified to account for these findings. Specifically, it was suggested that the introduction of addition would reduce the strength of zero associations and offer the possibility of $0 \times N = N$ errors. This suggestion is hard to evaluate as no accounts have been given of how addition and multiplication interact.

The suggestion of a separate mechanism, presumably using rules, for 0 and 1, is not without problems (section 2.1.3), but is attractive. The results from the cascade model, for example, are best when zero and one problems are excluded. No models have been offered to explain how such a dual-process account would work, and the task of building such a model seems large. Rather than do this, I suggest first further exploring the models we have. In particular, the mixing of addition with multiplication has been left almost untouched. Yet children learn multiplication only after learning addition, and addition appears to fuel zero and one errors. I propose the interactions between the two operations should be explored before separate mechanisms for certain operands are suggested.

3.5.3 *Verification and primed production tasks*

The settling models (IA, BSB, MATHNET) appear better suited to verification and priming tasks than the cascade model. The typical verification account (e.g., from BSB and IA) involves clamping input and output units and measuring the time required to reject or accept the suggested product. This apparently straight-forward way to model verification has a number of problems. First, it is necessary to only slightly activate the output layer. If the product to be tested was presented with full activation, surely the response mechanism should respond with that product immediately. The alternative is to delay the response in some way, but this has not been suggested by any of the models. It is not clear why the output layer should receive less activation from the presented problem than the inputs do. The motivation for this assumption is not clear from the IA account, and the BSB account does not clarify what assumptions it makes about product priming.

Figure 3.15 sketches one way in which the cascade model could be changed to capture verification. This speculative suggestion involves adding a separate network to the existing recall network. The verification network is presented with a product, while the recall network is presented with the two operands as usual. The hidden layer of the recall network forms part of the input to the verification network, and this should allow the system to verify the presented product. Note that a shared representation is presumed because it seems wasteful to suppose that the verification network requires a completely independent fact store.

The details of the verification system and the coupling to the recall network have not been worked out. For example, it is not clear how or why the verification network is trained, or if the training aids the recall network. Nor is it clear how control is allocated between the networks. However, it predicts that the verification network can be damaged, whilst the recall network remains intact. If the weights between the hidden layer and output layer of the recall network were independently damaged, it is possible that verification could proceed while production fails.

There is little evidence to support or refute the idea of verification process that is, in some interesting sense, separate from the recall process. However, tentative support comes from the study of a single brain-damaged subject by Dagenbach & McCloskey

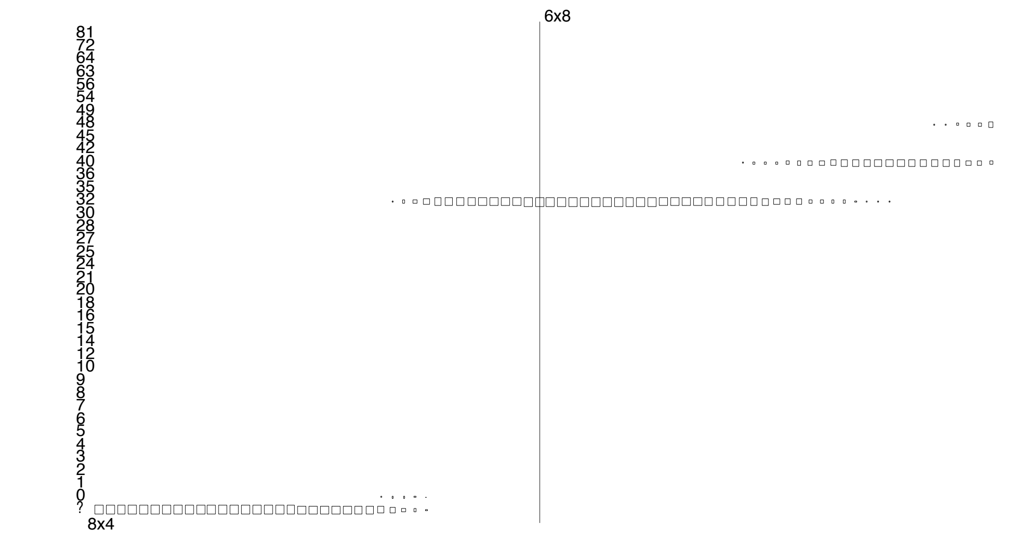

Figure 3.16. 8×4 followed by 6×8.

(1992). The subject was much better on the production task for subtraction than addition or multiplication. However, performance was comparable on all three operations for the verification task. Dagenbach & McCloskey "…suggest that the production/verification differences may be taken as support for the view that arithmetic fact retrieval processes are not the same in production and verification tasks" (p. 20).

Support for a separate mechanism presents a second problem for the settling models' account of verification. As verification is perceived as the product of a bidirectional settling process, why should damage selectively alter performance on just verification or just production? For the cascade model, a separate mechanism seems necessary to even begin simulations of the verification task.

Primed production is not so easy to model with the cascade system. The task requires some way to prime particular products before a problem is presented. Although output units could be activated before recall, it is not clear how this can be done without triggering the response mechanism (as described above). However, it is possible to show that the cascade model is sensitive to the context in which a problem is presented. In figure 3.16, the system has been presented with 8×4. After 40 processing cycles the input was changed to 6×8 without resetting the network to the "don't know" state. The vertical line in the centre of the figure shows the point at which the input was changed. The network

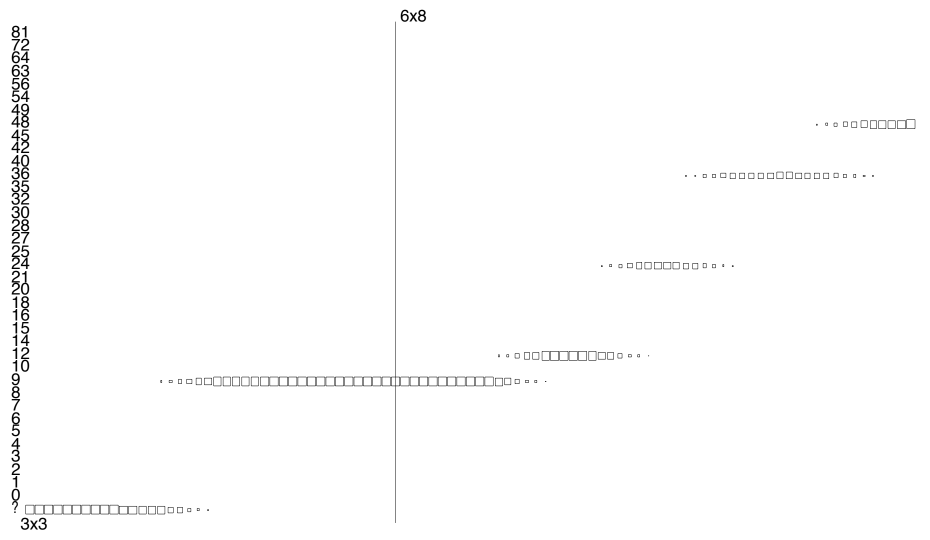

Figure 3.17. 3×3 followed by 6×8.

required 40 steps to reach threshold for the problem 6×8, stepping up through the 8 times table to reach the correct product.

Now consider figure 3.17. Here the first problem was 3×3, and the second remained at 6×8. The simulation shows clear differences in the processing, and 49 steps were required for an output unit to reach threshold on the 6×8 problem. The simulation had to ignore the fact that the previous product was above threshold when processing of the next problem began. This difficulty may be avoided if it is assumed that the activity of the system decays, to some degree, towards the zero-state before another product is presented.

These simulations also suggest a way for the model to produce different responses for a particular problem. Previous simulations have been deterministic: the response to a particular problem is always the same after training is complete. This is, in part, because the state of the system is reset at the start of processing to the "don't know" state. If the system is not reset, different responses will be produced. This context sensitivity is another possibility to be pursued.

3.6 Summary

The cascade model can capture aspects of RTs and errors of adults recalling multiplication facts. This chapter has outlined the assumptions required to do this, including assumptions about problem frequency and input representation. An initial analysis of the system has been presented, and variations on the model have been explored in an attempt to identify the factors that determine the behaviour of the system. In addition, simulations have been run to address the issue of zero and ones problems, and of network damage.

The cascade model requires further work:

- The RT results could better match the human results. With a one-of-N encoding, good RT curves are produced, but the errors are not so well distributed.
- Tie problems are solved by an ad hoc inclusion of a tie flag.
- The model does not capture the phenomena associated with zero and ones problems.
- Non-table errors are not explained.
- The verification and primed-production tasks are not accounted for.

However, an attempt has been made to present an explicit set of simulations that can be easily appraised. This is in contrast to other poorly-specified connectionist models, with the exception of MATHNET. Construction of the model has lead to a number of suggestions, in particular concerning the nature of false associations and coarse encoding, and of the verification task and zero and ones problems.

All of the connectionist models discussed use differing methods to measure RT, different input or output representations, different learning rules, training sets, hidden units, and so on, yet all show some degree of match to human behaviour. It seems that any kind of associative structure, with appropriate parameter settings, will fit the human empirical phenomena. This is encouraging because it seems that associative networks are just the right tools to model arithmetic fact recall. It is also discouraging because it hampers the discovery of the features that are responsible for determining the behaviour of human systems.

Plenty of important differences exist between the systems. The continued development of each of the models holds the promise of identifying those aspects that are pertinent to understanding human multiplication abilities—and weeding out those that are not.

As mentioned above, the cascade, MATHNET and IA models have been developed independently. One useful direction for this work is to look at the assumptions of the other models, and to see how they effect the performance of the cascade system. This could be profitable for discovering combinations of features that are sufficient to account for human behaviour. For example, although a tens and units output representation was originally used in the cascade model, it did not capture the distribution of errors found for adults. However, the model has developed since then, and it could be that the negative result was due to other parameters, such as the input representation. On the other hand, the representation of products may be a necessary feature of the model.

Throughout the chapter various ideas have been proposed as directions for future work. Some of the quickest extensions to the model include: adding a tens and units read out mechanism; detailing and training the verification network; testing the effect of various parameters, such as the number of hidden units or training times. Longer term goals include: exploring the relationship between addition and multiplication; studies of the effect of changes to the input representation, such as logarithmic input encodings; studies of pre-multiplication number understanding; systematic studies of network damage. However, the most pressing need is for a further systematic analysis of the existing networks: why are 9s so easy to recall? What are the effects of changes to problem frequency? What happens to the network as the weights develop during training? What are the effects of changes to the distribution of products? How would systematic changes to the sharpness of the input encoding alter the results? An understanding of the influences that lead to successful models may help develop an understanding of human arithmetic skills.

Part II

Multicolumn Arithmetic

CHAPTER 4

Symbolic Accounts of Arithmetic

Different tasks lend themselves to different styles of representation (Sloman 1985). As a rule of thumb it makes sense to use the most appropriate technology to model the phenomena of interest. An example would be to use connectionism for low-level processes (e.g., motor control), but switch to symbolic systems for higher-level tasks, such as planning (Clark 1989). Such hybrid views of cognition are attractive (Thornton 1991, 1992a; Rose 1991; Hendler 1989), but how do we know which technology is most appropriate for a given task?

This chapter reviews the symbolic models that have been built to capture the way children learn multicolumn arithmetic. Solving problems like 32—27 or 49×12 involves a host of skills: not only do you need to know the arithmetic facts, you also need to know how to borrow and carry, which column to process next, what to do when a number does not have a number below it (as in 12+5), and so on.

It appears that students are following rules when solving multicolumn problems. Perhaps the main evidence for this comes from the observation that children discover faulty rules (malrules) when learning arithmetic. Hence, it seems appropriate to use something like a production system to model multicolumn arithmetic. Indeed, to date the only systems used to investigate arithmetic have been rule-based systems (Brown & Burton 1978; Young & O'Shea 1981; Brown & VanLehn 1980; VanLehn 1983, 1990).

This chapter first looks at the kinds of mistakes children make when solving multiplication problems. The style of symbolic modelling is then described by briefly discussing a production system for multiplication. Section 4.2.2 considers the Young & O'Shea (1981) approach to modelling children's errors, and section 4.2.3 looks at the way VanLehn (1990) has modelled the errors. The conclusion is that the symbolic models offer a very plausible interpretation of children's arithmetic. Nevertheless, section 4.3 argues that there are good reasons for looking into a connectionist account of the phenomena. In particular, I aim to show that connectionism may be an "appropriate technology" for this domain, despite the phenomena's rule-like appearance. Many of the assumptions and ideas from the VanLehn and Young & O'Shea accounts are taken on board in chapter 5 which describes a connectionist model of multicolumn arithmetic.

4.1 Bug phenomena

At school children are introduced to multicolumn arithmetic over a number of years in lessons of ever-increasing difficulty. Maths textbooks (Howell, Walker & Fletcher 1979) suggests the following problem ordering for addition:

1. One column addition, sum < 10.
2. Two digit numbers, no carrying.
3. Addition of three rows.
4. Addition with gaps.
5. Two digits with carry.
6. Three column addition, without carry.
7. Three column with carry.

There is a similar sequence for multiplication, and VanLehn (1990, p. 13) identifies one for subtraction. By working through these stages students build up a hierarchy of maths skills, enabling them to tackle the next level of problems. Interestingly, Resnick & Ford (1982, p. 55) discuss studies involving a "deep end" approach to arithmetic teaching, in which students are just taught the higher level skills. The reasoning behind this idea is that the student should be given meaningful problems, rather than many boring, relatively simple problems. Of course, students taught this way still end up learning all the prerequisite skills. However, the majority of students seem to learn best when led through a hierarchy of skills, and this is the way arithmetic is taught in most schools.

Each lesson typically begins with the teacher working an example, and then the children solve similar problems on their own. Textbooks work in much the same way. An example is worked, by printing a snapshot of the various stages that need to be completed, and then there is a list of exercises.

There are at least two multiplication algorithms that are taught in schools. The one considered here involves building up the product of a multiplier on one row. For example:

$$\begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 8 \end{array} \qquad \begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 4\ 8 \end{array} \qquad \begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 4\ 8 \\ 0 \end{array} \qquad \begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 4\ 8 \\ 6\ 0 \end{array} \qquad \begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 4\ 8 \\ 3\ 6\ 0 \end{array} \qquad \begin{array}{r} 1\ 2 \\ \times 3\ 4 \\ \hline 4\ 8 \\ 3\ 6\ 0 \\ \hline 4_{1}\ 0\ 8 \end{array}$$

Studies of errors in arithmetic (Brown & Burton 1978; Burton 1982; Brown & VanLehn 1980) have characterized students as having buggy procedures—perturbations to the correct procedure. For example, one error for multiplication (zero-in-first-row) begins like this:

$$\begin{array}{r} 4\ 3\ 6 \\ \times \quad 5\ 1 \\ \hline 4\ 3\ 6\ 0 \end{array}$$

The student has incorrectly inserted zero into the first column. Another error is multiplies-using-addition-pattern:

$$\begin{array}{r} 3\ 4 \\ \times 1\ 2 \\ \hline 3\ 8 \end{array}$$

Here the student has multiplied each digit in the lower row by the digit above it in the upper row. These kinds of errors are quite different from the recall errors discussed in previous chapters. Rather than slipping and incorrectly following the correct arithmetic procedure, students are systematically using faulty rules. VanLehn (1981) reports that students exhibiting a particular bug can reproduce that bug a week later, giving exactly

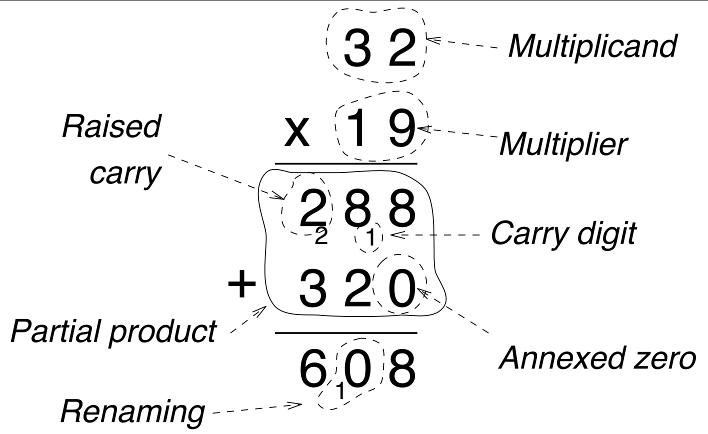

Figure 4.1. Terminology used for multiplication problems. For two row addition, the problem numbers are referred to as the "addends".

the same mistakes, digit for digit. However, there are limits to this stability, and this is discussed below.

Many different kinds of bugs have been described, some frequent, some rare. The most specific bug catalogues previously published have been for subtraction. I have decided to look at long multiplication and addition, so appendix A and B lists 39 addition and 63 multiplication bugs found in a trawl of the literature. Details of any particular bug, such as zero-in-first-row, can be found in these appendices. The descriptions use the terminology introduced in figure 4.1.

There is a great deal of interpretation in classifying a given mistake as being one bug rather than another. For example, if the child makes the mistake 6×0=6 as part of a problem, is that the bug N×0=N, or copies-multiplicand, or even adds-using-multiplication-pattern? Another problem is deciding that a given error is systematic. Different authors tend to take different approaches. The appendices are the result of combining the bugs listed by three authors, so at one level the problem of interpretation was solved because the authors had already classified the bugs. However, as discussed in appendix A, the interpretation problem then becomes one of finding corresponding bugs between the authors. The controversy surrounding bug categorization is not pursued further here, and it is assumed that at least some of the errors made by children are well described by buggy rules.

A number of studies have been conducted to discover the frequency of bugs. Of the 1147 seven- and eight-year-old children from VanLehn's Southbay study, 33 per cent harboured bugs, and a total of 134 distinct bugs were identified. Individuals may have more than one bug, and their set of bugs will change over time. Cox (1974) found that around 56 per cent of the 564 grade 1–6 children in her study made systematic errors in addition, multiplication and division. However, 10 percent were making "random errors", and the remaining children were error-free or just slipping. VanLehn (1981, p. 6) also found that 10 per cent of grade 3 children's error could not be analysed in his study of subtraction. Still, at most 90 per cent of children's behaviour can, in principle, be explained by reference to a possibly faulty rule set.

4.2 Models

The formalization of the child's rule set is discussed in this section—including rule acquisition, operation and representation. Starting with the account of arithmetic developed by Young & O'Shea (1981), I hope to show how the phenomena can be neatly modelled by production systems. Repair theory (Brown & VanLehn 1980) appeared at about the same time as the Young & O'Shea model, and has since gone through many refinements to reach its current form as Sierra theory (VanLehn 1990). Despite a number of criticisms, Sierra theory is the best account we have of faulty rule acquisition. The reasons for this are described below.

4.2.1 A production system for multiplication

Young & O'Shea produced a production system to model correct and buggy subtraction. In the same style as the Young & O'Shea system a multiplication production system was built by Shaaron Ainsworth as part of her MSc work at Sussex. Table 4.1 lists the production rules for correct multiplication. In what follows, many of the implementation features of this particular system are glossed over, and it is assumed that the model works in much the same way as the Young & O'Shea system.

As is usual of production systems the left-hand side of the productions specify queries to be matched against a working memory. The right-hand side specify actions which can

	Conditions		*Actions*
`INTO:`	`[processmult]`	⇒	`readintandb();`
`SM:`	`[t ?t] [b ?b] [c ?c]`	⇒	`do_calc();`
`NX:`	`[next_top]`	⇒	`[processmult] shift_top_left();`
`WM:`	`[result ?u] [carry ?c]`	⇒	`writedown(); [next_top]`
`CC:`	`[no_more_top]`	⇒	`checkcarry(); [checkbottom] [addzero]`
`CB:`	`[checkbottom]`	⇒	`check_bottom();`
`FI:`	`[none_left]`	⇒	`[stop]`
`NB:`	`[no_more]`	⇒	`endmult(); [startadd]`
`CO:`	`[startadd]`	⇒	`readincolumn();`
`DA:`	`[column ?len ?dig]`	⇒	`do_add();`
`ML:`	`[next_left]`	⇒	`[startadd] moveleft();`
`WA:`	`[u ?u] [c ?c]`	⇒	`writeadd(); [next_left]`
`CA:`	`[no_more_digits]`	⇒	`checkadd();`
`AZ:`	`[addzero]`	⇒	`add_zero();`

Table 4.1. Production rules for correct multiplication.

be either calls to procedures (e.g., `do_calc()`) or items to deposit in the working memory (such as `[processmult]`, the initial goal). In working memory the partial solution is represented by a list containing: the two digits being multiplied; the result of the current row of multiplications; two numbers indicating which digits in the multiplicand and multiplier are being considered; and, the number currently being carried. For example, while solving the problem 34×22, the representation will be…

```
[ [3 4] [2 2] result=[6 8] topdigit=1 botdigit=2 carry=0 ]
```

…indicating that the digits just considered were 3 (`topdigit` is 1, the first number in `[3 4]`) and 2 (the second number in `[2 2]`). The result of the row is 68, with no carry. In the Young & O'Shea model there was no explicit representation of `topdigit` or `botdigit`, or their equivalents. Their production system used operations such as `ShiftLeft` and `NextColumn` to focus attention without worrying about the underlying registers. However, registers are required, and it is useful to have them explicitly shown.

Using this representation of the problem, and the `topdigit` and `botdigit` registers, the answer can be gradually built up in working memory. The procedure `readintandb`, for example, identifies the digits to be multiplied. `Shift_top_left` updates the `topdigit` and `botdigit` registers to focus on the next digits in the problem. Eventually, `shift_top_left` runs out of digits and deposits [no_more_top] in working memory. This then caused the productions system to enter into a different part of the arithmetic routine, ready to process the next multiplier.

The details of this model are not of great interest here, but note the style of processing. The model captures multiplication as small skill modules, which are pattern or event driven. For example, certain actions bring about the recognition of certain other facts about the problem, which leads to other rules firing as happened with [no_more_top]. Central to the model is the use of registers, such as topdigit, to keep track of the system's position in the problem. Registers are also used in the eye movement models of arithmetic proposed by Suppes, Cohen, Laddaga, Anliker & Floyd (1983), and will be used in the connectionist model described in the next chapter. Finally note that the skill has been described at a level equivalent to how one might describe multiplication verbally. Clearly there are other processes going on which are not modelled by the production system, such as eye movements, recall of arithmetic facts, physical action of writing, and so on. However, it seems that this level is adequate for capturing arithmetic bugs.

4.2.2 Modelling bugs—the Young & O'Shea way

Certain bugs can easily be modelled by removing a rule from the production system. For example, removing the AZ rule (from figure 4.1) produces the bug forgets-annex. Other bugs require that rules in the rulebase are replaced with faulty versions. For example, quits-after-first-multiplier is generated by replacing the rule CB with the following:

```
bugCB:   [checkbottom]   ⇒   [no_more]
```

Young & O'Shea modelled errors in subtraction in this way, by adding and omitting various rules. They discovered that a small number of changes could cover a large number of observed errors. As an example Young & O'Shea note (p. 163) that two rules (one correct, one faulty), in four possible permutations (each either present or absent from the rulebase) could account for 115 out of 124 errors found in their corpus, in addition to the correct algorithm.

To account for all the observed errors, Young & O'Shea included a number of other faulty rules. Clearly, it would be trivial to construct a production system to account for a particular error, and such a production system would be of no psychological use. However, an attempt was made to constrain the productions in various ways. Young & O'Shea's aim was to model a particular child with a particular production system. Different children will have different productions, and each child's set of rules will change over time. Given this constraint, it is still possible to build a production system "...whose conditions were so complex, bizarre, and ad hoc that each subtraction problem was effectively treated as a separate case..." (Young & O'Shea 1981, p. 164). This consideration applies to all cognitive models, and it is not possible to exactly specify a set of constraints to avoid ad hoc models. Young & O'Shea used three heuristics to guide their model building:

1. Adopt a particular style for the rules.

2. Avoid problem-specific symbols in the rules (i.e., numbers).

3. Minimize the number of changes between rulebases.

Using these heuristics, Young & O'Shea optimized a number of production systems to fit the errors found in their data. Starting with the correct production system for subtraction, rules were changed to improve a score. The score is the number of errors predicted by the system, minus the number of false errors. False errors are not errors which have never been observed (star bugs, discussed below), but are due to the fact that children are not fully consistent in applying faulty rules. It may be that a child exhibiting a particular error does not make the same mistake at every opportunity. A production system following a rule will always make the error. False errors are the mismatch between the child's and the system's error performance. Initially, the correct system makes no false errors, and predicts no bugs. Young & O'Shea mutated the rulebases and finished with eight production sets which accounted for 160 of the children's errors, missing 18 and falsely predicting another 32 (1981, table 3, p. 166).

It is interesting that arithmetic skills can be captured in this way, but there is no account of where the rules (correct or otherwise) come from. The modularity of productions means that learning can be thought of as the accumulation of rules (Young 1974; Neches, Langley & Klahr 1987), although no such account is given. The Young & O'Shea model provides snapshots of children at various stages of development, but does not discuss the transition between stages (acknowledged on p. 176). There are ways to incorporate learning processes into the Young & O'Shea "family" of models. Klahr (1992) lists a number of mechanisms, including condition generalization and discrimination, rule composition and chunking, proceduralization, and rule strengthening. There is a gradual nature to some of these mechanisms, such as rule strengthening, but not to all of them. At some stage new rules need to be added to the rule base (Klahr 1992, p. 170). This is what is meant by the phrase "snapshot account": at one moment the rule base contains a certain set of rules, and at the next is contains some new ones. The Young & O'Shea model is an extreme example, but the granularity of other models (including VanLehn's) means that they also qualify as snapshot accounts of development. This style of learning will be contrasted with the gradual learning performed by connectionism later in the chapter. Rather than discuss ways in which learning could be incorporated into the Young & O'Shea model, attention will be focussed on VanLehn's learning system in section 4.2.3.

Inconsistent rule following is not accounted for by the Young & O'Shea model. The term "bug migration" (Brown & VanLehn 1980) refers to the phenomenon of children switching between sets of bugs. Some bugs are persistent, lasting for years, while others come and go over a short period to time—even during a single test. Hennessy (1990) believes that there is much more instability in children's performance than has been previously suggested. She argues (pp. 178–180) that a more accurate, and less subjective means of bug diagnosis is required. The Cox (1974) study defined a bug as an incorrect procedure observed on three out of five problems. This kind of definition leads to the three error categories of slip, bug and undiagnosed. VanLehn (1981) uses a different, more complicated method, designed to "mimic the intuitions of human diagnosticians" (p. 15). A student is said to have a particular bug if the diagnosis makes "enough" true predictions, and at least more true predictions than false predictions. "Enough" is defined by a number of conditions, e.g., 75 per cent of predictions must be true (for further discussion on the problems of diagnosis, see Brown & Burton 1978; Burton 1982). For Hennessy, any faulty procedure counts as a bug, no matter how frequent it is, with the exception of slips. This definition clearly eliminates the "undiagnosed" category of errors, but increases the instability associated with a given student.

A number of studies have attempted to pin-down the extend of bug migration. In a follow-up study, one year after her original study, Cox (1974) found that 25 per cent (16 of 64) students exhibited the same bug as in the previous year, and 17 per cent (11 of 64) were showing a different error over all the arithmetic operations. VanLehn (1981, 1990) found that in the short term (over 2 days) only 9 per cent (3 of 32) students produced the same errors, where as 38 per cent (12 of 32) exhibited bug migration for subtraction only. The different ages of the children, different experimental procedures, and small sample sizes makes it difficult to compare the Cox and VanLehn results on bug migration.

Despite the problems of diagnosis, it is apparent that bug migration is a common phenomenon. Migration is usually defined as changes occurring "without intervening instruction" (VanLehn 1990, p. 54), but this does not exclude the possibility that some kind of learning is causing migration. However, the Young & O'Shea model does not address bug migration or learning, and as such it fails to capture important aspects of children's arithmetic.

4.2.3 Modelling bugs—the VanLehn way

VanLehn's (1990) Sierra theory of learning arithmetic is the most detailed account of multicolumn arithmetic to date. Like Young & O'Shea, he focuses on long subtraction, but asserts that the principles should apply not only across the other three arithmetic procedures, but also to other domains. The account, a continuation of repair theory (Brown & VanLehn 1980), tackles two problems of bug migration and learning. Three elements of his theory stand out:

1. Children's mistakes are the result of various kinds of *syntactic* changes to rules. This not only includes the faulty rules from the Young & O'Shea account, but also run time changes.

2. Faulty rules are due to the *skew* in the school curriculum.

3. Learning is from *examples*, not verbal recipes.

Several observations support these points. First, it seems that students do not understand the operations they are performing: they are symbol-pushing (VanLehn 1990, pp. 38–40). There is now work considering the role of the semantics of arithmetic (Hennessy 1990; Payne & Squibb 1990; Ohlsson & Rees 1991) but there are a number of reasons why it seems appropriate to just model syntax. To support his "ateleological assumption", VanLehn points out that students who understand the arithmetic procedure they are trying to perform should be able to see if their procedure is wrong, and also be able to fix it. Intuitively it is clear that students can learn procedures without the slightest understanding of the underlying principles, and this point appears uncontroversial (VanLehn 1990, p. 38).

The work of Hughes (1983) also suggests that children have trouble with the syntax of arithmetic:

> The problem is not that young children are completely lacking in their number concepts…Rather the problem is that they are encountering a novel code, or representation system, which may be like a foreign language to them (p. 209).

In other words, school arithmetic is, as Donaldson (1978) calls it, "unembedded". This is demonstrated by the following protocol from a four year old reported by Hughes (1983, p. 211):

Adult:	How many is two and one more?
Child:	Four.
Adult:	Well, how many is two *lollipops* and one more?
Child:	Three.
Adult:	How many is two *elephants* and one more?
Child:	Three.
Adult:	How many is two *giraffes* and one more?
Child:	Three.
Adult:	So how many is *two* and one more?
Child:	(looks adult straight in the eye) Six.

These problems with syntax should come as no surprise given the way arithmetic is taught. VanLehn (1983, p. 82) points out that arithmetic texts are nothing like cookbooks or other kinds of manuals: they consist mostly of worked examples and exercises. In contrast to examples are "explanations"—some form of natural language information, perhaps given by the teacher in class. VanLehn argues (1990, pp. 96–103) that arithmetic is primarily learned from examples, and not explanation. His justification of this assumption first notes that AI has experienced a number of problems in translating from natural language to programs. However, there has been much more success in learning from examples. Second, VanLehn supposes that if children learned mostly via natural language, there should be language fragments in their bugs, such as references to the "tens place" or the "multiplicand". However, VanLehn found that "…85 percent of all the observed bugs can be described with a small set of visual/spatial features. No bug requires linguistic features for its description" (p. 102).

So it seems that children learn the steps in arithmetic procedure by example, without an understanding of the algorithm. But how do they acquire these procedures?

Learning by induction

Like Young & O'Shea, VanLehn proposes that children have buggy core procedures. However, unlike Young & O'Shea, VanLehn shows how these procedures can be induced from the examples given in a lesson. This section describes the learning part of Sierra, the name of VanLehn's model. In general terms, there is a learner which takes a lesson, L_i, and the current state of a student, P_{i-1}, and returns a set of procedures that are consistent with the examples in the lesson P_i^1, P_i^2, …, P_i^k. That is, the learner returns a number of rule sets, some with bugs, and some without. Each of the procedures can be tested to see how its bugs compare to those of children. Then another lesson is administered, and the learner generates procedures that can solve harder problems.

Much of VanLehn's (1990) work is concerned with ways to constrain the learner to induce just those things that humans learn. Hence, a number of assumptions are laid out, including:

- New rules are assimilated with the old. That is, the rules of P_{i-1} are a subset of those of P_i.
- One disjunct per lesson. At most one subprocedure (a new branch point) can be introduced to a rule in a lesson.
- The most specific patterns are induced.

$\text{Subtract} \rightarrow \text{Sub1Col}_C \text{(Foreach C)}$
$\text{Sub1Col}_C \rightarrow \text{Diff}_C$
$\text{Sub1Col}_C \rightarrow \text{BorrowFrom}_{C+1}\text{Add10}_C\text{Diff}_C$
$\text{BorrowFrom}_C \rightarrow \text{Decr}_C$
$\text{BorrowFrom}_C \rightarrow \text{BorrowFrom}_{C+1}\text{Add10}_C\ \text{Decr}_C$

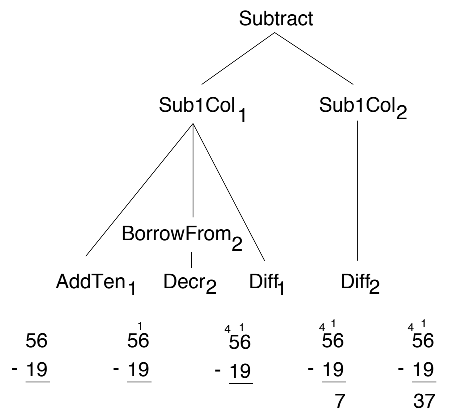

Figure 4.2. An example of parsing a subtraction example (from VanLehn 1990, pp. 125 and 136). Using the rewrite rules from the top of the figure, the stages in the examples at the bottom of the figure can be represented as a tree structure. Note that the rules are context sensitive—i.e., $\text{Sub1Col}_C \rightarrow \text{Diff}_C$ is only applied when the top digit is $\geq$ bottom digit. The subscript (C) indicates the column in which the operation applies.

Using a rule representation it is possible to parse arithmetic examples (see figure 4.2). For problems that the system cannot solve, which are often the problems in the next lesson, it will not be possible to produce a parse tree. In this situation VanLehn applies top-down and bottom-up parsing to construct as much of the parse as possible. However, the resulting parse with have a gap in it where the child nodes do not meet with parent nodes. VanLehn then uses an algorithm called "parse completion" to fill in the gaps.

For the problems that cannot be solved, there will be a large number of partial parses. Top-down and bottom-up parsing is used to produce a set of these "skeletons"—the parent and child nodes that define the gap in a tree. There will be at least one skeleton from each example that could not be parsed. Because of all the constraints, the intersection of all the skeletons can produce one unique skeleton. If this is not the case, the algorithm uses various biases to select a single skeleton. For example, one bias is to select the skeleton with the lowest parent nodes.

In developing the constrains on the learner, VanLehn stresses the importance of a well formed lesson sequence. Convention allows the learner to assume that lessons will obey certain rules (VanLehn 1987). The lesson should introduce just one "knowledge chunk" (disjunction or subprocedure), and this will often allow the learner to solve problems that it could not solve before the lesson. VanLehn suggests that "…experienced teachers generate such lesson sequences naturally, without even realizing that their lesson sequences obey the two conventions" (1987, p. 7). In short, good teachers give lessons that make the learner's task easier.

The skeleton defines the goals and subgoals of a rule. Patterns also need to be learned to clarify when the rule is to be applied. For this, VanLehn uses Mitchell's (1977) version spaces (see also Thornton 1992b, chapter 2). Each example arithmetic problem is defined by a set of position, fact and action primitives. Patterns may contain factual primitives, such as "less-than" or "Zero?". Just three actions are available: writing a character, erasing a character and overwriting a character with some other character. See table 4.2 for an example procedure and problem representation. These primitives were selected by guess work, and a more informed approach would require a theory of perception.

Of all the patterns that could be induced, the most specific pattern is the one that is learned. For example, from the subtraction shown in figure 4.3b, the following rule will be learned:

```
borrow column = leftmost and left-adjacent
```

This is the most specific pattern to describe the borrowing column: the column that is both leftmost on the page and also adjacent to the current column. Most specific is defined relative to the granularity of the representation language, hence ensuring that generalization takes place, and the learner does not simply acquire a list of past problems. There are other technical reasons for why this constraint is proposed. When combined with the constraint that patterns can only contain conjunctive connectives, it can be shown that the version space is finite, and there exists a fast algorithm for computing the whole version space (VanLehn 1990, p. 152). From experimentation with the system, VanLehn discovered that the most specific rules learned are sufficient for generating impasses.

VanLehn (1990, chapter 6) describes a number of other restrictions on pattern learning.

`(Problem 1)`	*Object 1 is a subtraction problem*
`(Column 2) (Column 3)`	*Objects 2 & 3 are columns*
`(Part 1 2) (Part 1 3)`	*The columns are part of the problem*
`(First 1 3)`	*Object 3 is the leftmost object*
`(Adjacent 1 2 3)`	*The columns are adjacent*
`(Cell 4) (Cell 5) (Cell 6)`	*Objects 4–5 are cells*
`(Digit 4) (Digit 5)`	*Objects 4 and 5 are digits*
`(Blank 6)`	*Object 6 is a blank cell*

```
Sub1Col(C) OR
1. [And (Digit T) (Part-of T C) (First T C)
        (Digit B) (Part-of B C) (Middle B C)
        (Ordered C T B) (Adjacent C T B)
        (Value-of TV T) (Value-of BV B) (LessThan TV BV)
   ->   (Borrow C)

2. [And (Digit T) (Part-of T C) (First T C)
        (Digit B) (Part-of B C) (Middle B C)
        (Ordered C T B) (Adjacent C TB)
        (Value-of TV T) (Value-of BV B)
        (Less-Than-or-Equal BV TV)
   ->   (Diff C)

Diff(c) AND
1. [And (Digit T) (Part-of T C) (First T C)
        (Digit B) (Part-of B C) (Middle B C)
        (Cell A) (Part-of A C) (Last A C)
        (Ordered C T B) (Adjacent C T B) (Ordered C BA)
        (Value-of TV T) (Value-of BV B)
        (AbsoluteDifference TV BV AV)
   ->   (Write AV A)
```

Table 4.2. Part of the problem representation for 57—9, and part of a subtraction procedure, both using Sierra's representation from (VanLehn 1990, table 3.8 and 3.10).

Indeed, many of the assumptions have been left out, and much of the detail skipped over in this review. In particular the justifications for many of the assumptions have been omitted. In essence, though, the learner is a highly constrained inductive mechanism, learning control structures to parse problems and patterns in terms of a predefined set of primitives. The constrains are psychologically motivated, and strong enough to make the learning problem tractable.

The impasse-repair process

In addition to the learner, the other component of Sierra is the solver. This part of the model applies the learned rules to specific problems. Young & O'Shea built their rulebases in such a way as to ensure that the condition patterns would be appropriate for all situations after conflict resolution. Building on the work of Brown & VanLehn (1980), Sierra incorporates the notion of an *impasse*. For example, when no single rule is uniquely specified, and impasse is said to have occurred, and Sierra *repairs* the impasse with a number of local modifications to the solver's state. That particular case, where more than one rule matches, is an instance of a decision impasse, and appears to be just another kind of conflict resolution. In fact VanLehn suggests that there are three kinds of impasse:

1. Decision, when a decision is needed, but cannot be made.
2. Reference, for an object in a pattern that is not uniquely specified.
3. Primitive, where some primitive operation cannot be carried out—e.g., a student trying to solve 0-1, but not knowing the answer.

These three kinds of impasse are all that Sierra needs to model children's errors in subtraction.

When an impasse occurs, the local problem solver has to make a repair to the current state so that Sierra can continue with the problem. VanLehn identifies three kinds of repairs:

1. No-op, which simply skips the offending operation.
2. Barge-on, by relaxing the conditions and then applying the rule.
3. Back-up, by going back and changing a previous action.

Again, just these three kinds of repair are needed to model subtraction and other problems (VanLehn 1990, p. 43). The "Cartesian product" of repairs and impasses (each repair applied to each impasse) should be the repair strategies observed in the bug data. This assumes that repair strategies are not limited to specific impasses, but are general methods that could be applied to many different impasses. VanLehn (1990, pp. 44–54) concludes that although there are some biases favouring particular repairs for particular bugs, repair selection can be approximated by random choice. This suggests a criterion

$$\begin{array}{r} \overset{2}{\cancel{3}}\;6\;\overset{1}{5} \\ -1\;0\;9 \\ \hline 1\;6\;6 \\ \text{(a)} \end{array} \qquad \begin{array}{r} \overset{5}{\cancel{6}}\;\overset{1}{5} \\ -2\;9 \\ \hline 3\;6 \\ \text{(b)} \end{array}$$

Figure 4.3. The bug always-borrows-left (a), where the student borrows from the leftmost column. This behaviour is appropriate for two column problems, such as (b).

for deciding whether or not to include new kinds of impasses or repairs: the new impasse or repair, when multiplied in with the other impasses and repairs should predict plausible bugs and no implausible bugs. Indeed, VanLehn reports (p. 53) that when impasse-repair independence was first tested, it predicted 16 new bugs, 7 of which have been found.

As an example of the impasse-repair process, consider the subtraction bug always-borrows-left, shown in figure 4.3a. As noted above, the skew in the curriculum means that children are exposed to two column problems before three column problems. Given the learner's bias to learn the most specific patterns, it acquires the rule "borrow from the column that is leftmost and also left-adjacent". In the context of two column problems this is an appropriate rule (see figure 4.3b). However, when the child encounters a three column problem which requires borrowing for the first column, an impasse occurs: there is no column that is both leftmost and left-adjacent. At this point a local problem solver takes control from the interpreter and applies a repair. If the barge-on repair is used, one of the rule's conditions is relaxed. The bug shown in figure 4.3a results from relaxing the "left-adjacent" requirement; relaxing "leftmost" clause results in the correct solution for the subtraction.

Empirical adequacy

By the Sierra account, bug migration is the result of applying different repairs to the same impasse. However, the local problem solver may construct a *patch* to the buggy procedure. This associates the current repair with the current impasse, so that when the impasse occurs again, the same repair will be used. In this way both bug migration and stable bugs are accounted for.

The rule sets produced by Sierra (the predictions of the procedures that children could learn) are tested for bugs. After each lesson, the results from the learner are passed to the solver. VanLehn (1990, p. 28) claims that "…many bugs are caused by testing beyond training…". Hence, the system is tested not only on the current lesson, but also on harder problems not taught by the current lesson. Buggy behaviour can also be found on the current test, especially if the deletion operation has been applied to a rule set (p. 106). This removes the most recently added goal from a procedure, resulting in procedures that exhibit bugs such as does-not-carry-over-blank, stutter-add and does-not-carry. Allowing rules other than the most recent one to be deleted may result in implausible bugs, e.g., a rule set without the rules for writing down the answer.

The results from the solver are diagnosed as buggy or not by passing them to an automated diagnosis tool called Debuggy (Burton 1982). The overlap between observed bugs and predicted bugs is the criteria by which Sierra is evaluated. It may be that Sierra predicts more bugs than have been observed. This is not a problem providing that the unobserved bugs are not implausible (star bugs). VanLehn comments (p. 19):

> When Sierra generates a star bug, it is missing some kind of constraint. Star bugs indicate that the theory needs revision. So avoiding the generation of star bugs is just as important as generating observed bugs.

In the Southbay study of 1147 test solutions, 75 individual bugs were observed. Of these, Sierra predicted 28, and failed to predict 47. Sierra also predicted 21 plausible bugs which have not yet been observed, and 7 star bugs. VanLehn argues (p. 200) that when the simulation catches up with the theory, 39 of the 75 bugs will be predicted (missing 36), with no star bugs and just 21 plausible (but unobserved) new bugs. Most of the 36 bugs that are not predicted are pattern errors (e.g., N−0=0). If Sierra could generate an impasse when there is a zero to be borrowed from or to, then these bugs could be explained. However, VanLehn notes that "…empirical quality is not the only measure of theoretical validity. It must be balanced against explanatory adequacy…" (1990, p. 205). Young & O'Shea explain pattern errors as being derived from confusions from other operations (i.e., N+0=0). It is a simple matter to write a production rule with a condition to capture such pattern errors. VanLehn is critical of this method, and demands an explanation of why such errors only occur in the context of zero and not other numbers. "If the model is too easily tailored, then it is the theorist and not the theory that is doing the explaining" (VanLehn 1990, p. 204).

4.2.4 *Summary*

The systematic mistakes made by children solving arithmetic problems suggests that production systems should be used as a model. Young & O'Shea (1981) built such a model for subtraction, and demonstrated that errors could be modelled by small perturbations to the rule set for correct subtraction. The modular nature of the system also suggests that learning could be modelled by adding more rules to the rulebase. However, the Young & O'Shea model did not account for learning.

Repair theory (Brown & VanLehn 1980) suggests that buggy core procedures cause impasses which are repaired from a set of independent repair heuristics. This principle avoids Young & O'Shea's arbitrarily hand-coded malrules, and can also account for bug migration. In fact repair theory predicted bug migration before it was found (VanLehn 1983; VanLehn 1981). Sierra theory (VanLehn 1990) developed from repair theory, and includes a learning component. Buggy rules are induced from a skewed curriculum and then interpreted by a problem solver which detects and repairs impasses. The constraints on the theory are argued for with evidence from psychology, and ensure that the learning task is not only tractable, but also fits the bug data rather well.

The difference between core procedure bugs and impasses is that the latter are detectable by the person following the procedure. However, VanLehn (1990) notes that the existence of impasses is an empirical issue: ACT* (J. R. Anderson 1983) doesn't require impasses, but Soar (Laird, Newell & Rosenbloom 1986) does. A discussion of the importance of impasses and the relationship between Soar and Sierra is deferred until section 5.6.1. Despite the subjective nature of labeling errors as bugs or slips, or as plausible or implausible, it seems that rule based systems are adequate for modelling children's arithmetic.

4.3 Why connectionism?

Sierra theory is a success story of symbolic modelling. Admittedly it is not without some problems, but it is the most significant existing account of children's arithmetic and of faulty rule acquisition in general. Arithmetic looks symbolic and can be modelled with symbolic systems, so why bother thinking about a connectionist model? There are at least five observations which suggest that connectionism is worth considering:

1. The construction of a connectionist model of arithmetic is a tough engineering task, but only by implementation do we fully realize the difficulties faced by connectionism and how they might be solved.

2. There is support for the idea that connectionist systems are the appropriate tool for capturing developmental phenomena.

3. More generally, connectionist system exhibit mind-like properties, such as automatic generalization and graceful degradation. Connectionist computation is "brain-style" computation (Rumelhart & McClelland 1986b).

4. Theory and implementation are never as independent as one would wish. Without an alternative for comparison there is a danger that Sierra is unduly biased by symbolic AI.

5. Connectionism is changing our understanding of notions like "symbol".

Chapter 5 gives details on the construction of the connectionist model of multicolumn arithmetic. This section expands on the remaining (strongly interrelated) points.

4.3.1 Development

There is a growing body of research applying connectionism to problems in developmental psychology. Examples include general treatments of developmental phenomena, such as stages of development (Shultz 1991), and models for: balance scale (Shultz & Schmidt 1991; McClelland 1990); seriation (Mareschal & Shultz 1993); English verb morphology (Plunkett & Sinha 1991; Rumelhart & McClelland 1986a); and concept formation and vocabulary growth (Plunkett & Sinha 1991). In addition there are a number of models which look at the importance of development in terms of constraints which help the learner (most notably Elman 1991, 1989a). Some authors assert that "…PDP models provide a superior account of developmental phenomena than that offered by cognitivist (symbolic) computational theories" (Plunkett & Sinha 1991, p. 1). Given this interest in development, it is natural that connectionism should be applied to arithmetic skills.

It is worth briefly looking at a couple of these models to appreciate the connectionist approach to development. Shultz (1991) comments that the vast majority of developmental studies have focussed on what has developed rather than on how transitions occur. For example, four stages have been identified for the balance scale task. In these experiments the subject (usually a child) is presented with a beam balance. Various weights are placed at various distances from the fulcrum. Typically the subject is asked to judge which side of the balance will go down when a weight is removed. In the first stage the subject uses weight alone is used to decide how the scale balances. Later, distance from the fulcrum is correctly used, but only when the weights are equal. Stage three is characterized by the correct use of weight and distance in most instances, but confusions occur when one side has the greater weight and the other side has the greater distance. Finally, the stage four subject multiplies distance by weight and compares each side's product to find the answer.

As a starting point to understanding the transitions between the stages, Shultz identified a set of essential features of stages based on a previous analysis of Piaget's work. These features included notions of qualitative change, stage ordering, and denied any abruptness in transitions. To elaborate, changes between stages seem to involve a qualitative, and not quantitative, change. Transition is "…not simply a matter of adding more information, but rather the emergence of a substantially different way of processing information" (Shultz 1991, p. 105). Stages also tend to be acquired in a particular order, although stage skipping and regression are occasional observed. The transition from one stage to another is more gradual than abrupt. That is, signs of stage $n + 1$ performance are present during stage n, and perfection at stage $n + 1$ is not achieved until the end of stage $n + 1$.

Shultz listed four ways in which stage development can occur in connectionist networks.

1. *Hidden unit herding.* For multilayered networks (e.g., those trained with backpropagation), each hidden unit does not select a unique role early in learning. Rather, all the hidden units move to reduce the current largest error (Fahlman & Lebiere 1990). Eventually hidden unit responsibilities are sorted out, but in the intervening time stages can be observed. An example of this occurs in the learning of past tenses (Rumelhart & McClelland 1986a; Plunkett & Marchman 1990; Marchman 1992).

2. *Over generalization.* Hidden unit herding is one form of over generalization, but a network without hidden units can also over-generalize. An initial period of over generalization could be seen as one stage, with later stages occurring as the network learned the fine distinctions in the training set.

3. *Training bias.* As seen in chapters 2 and 3, networks are sensitive to problem frequency. By manipulating the training environment, networks can be made to exhibit stages.

4. *Hidden unit recruitment.* Construction algorithm, such as cascade correlation (Fahlman & Lebiere 1990; Fahlman 1991), incrementally install new hidden units to reduce error. These kinds of networks are not pursued further here, but Shultz notes that of 24 hidden units recruited in his model of the balance scale task, 13 corresponded to stage progressions.

These kinds of connectionist stage transitions also have other properties. For example, qualitative changes in behaviour are observed, although the weights of the networks only go through small quantitative changes. It is noted, though, that whether a change is quantitative or qualitative often depends on how closely it is looked at. Likewise, the grain size of a particular model will determine the degree to which changes are more or less qualitative. In connectionist models it is also found that stage transitions are tentative at first, with the network bobbing between stages before committing itself. These behaviours are found in both the cascade correlation (Shultz & Schmidt 1991) and the backpropagation (McClelland 1990) models of the balance scale task, and the model of seriation (Mareschal & Shultz 1993). The conclusion drawn is that "…connectionist modelling of stages is so far quite consistent with the major regularities in the psychological literature on cognitive development" (Shultz 1991, p. 109).

Elman (1991) has looked at the significants of certain kinds of development for the learner. Elman notes that if a child is given just positive examples of some data, then only a regular grammar can be learned. However, natural language appears to belong to a more complex class of grammars. Given that children do learn language, the usual assumption is that there is something innate which constrains the learner to allow it to learn natural language. Elman uses the fact that the learner develops as a constraint which enables it to acquire a complex grammar. In one experiment, a network's memory was allowed to grow over time. Specifically, a simple recurrent network (see figure 4.4) had its memory "blanked" every third or fourth word. This was done by resetting the context layer every third or fourth input. The memory span was increased over time by blanking less often. The network was trained on a large, complex grammar which included number agreement, use of direct and optional arguments to verbs, and sentence embedding. Previously it had been established that a standard simple recurrent network could not learn this grammar. However, by starting with a small memory span and gradually increasing it, the grammar could be learned. The network's memory limitations advantageously constrained what could be learned, in effect reducing the size of the solution space. That is "…they [the memory limitations] act as a filter on the input, and focus learning on just that subset of facts which lay the foundations for future success" (Elman 1991, p. 8). This method only applies in structured environments, in which fragments of the problem are useful in solving the whole problem. These conditions are not met when learning a random set of facts. But Elman comments: "In practice, the world is not a random place, and the sorts of things children have to learn about typically contain a great deal of structure" (1991, p. 9).

Returning to arithmetic, it seems that the symbolic accounts have mainly been applied to the capture of the end-result of learning. Hennessy (1990, p. 175) pin-points the problem:

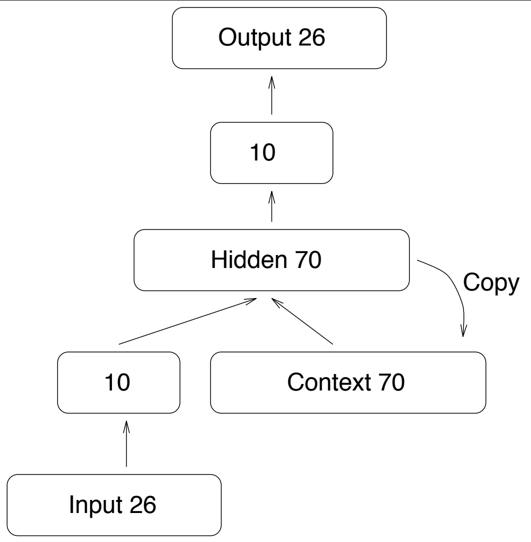

Figure 4.4. The simple recurrent network used by Elman (1991). Each word in the lexicon is represented as a 26 bit vector. The network learns to re-encode the input into 10 units. A sentence is fed one word at a time to the network.The task is to predict the next word in the sentence. At each time step the hidden units are activated and copied back to the context layer. Hence, the context layer acts as a memory of the network's previous state. As words are fed in, the hidden layer uses the current word and the context to build up a representation of the sentence.

> While the production-system framework has yielded a number of useful characterisations of children's procedural skills, it has failed so far as an attempt to model the process involved in development. Its focus is on describing what is learned—the endstate of a learning process or a snapshot view—rather than on how malrules are actually generated.

This applies to Sierra, where the learner takes a lesson and a rulebase, and returns a number of updated rulebases that are consistent with the lesson. There is no consideration of how the new rules are incorporated into the solver, or the effects this has on the system's performance. VanLehn is developing a theory of impasse-driven learning, but as discussed in section 5.6.1, this is not yet fully specified. An inescapable aspect of connectionism is the gradual nature of learning. There does not have to be "a moment" when learning "happens", and hence the snapshot problem of learning in the Young & O'Shea and the VanLehn models is side-stepped (Bates & Elman 1992, p. 14).

In summary, connectionism looks promising for understanding aspects of development. As a *metaphor*, connectionism does seem better suited to account for development.

Bates & Elman (1992) list the features of symbolic AI which they believe has hindered the understanding of development: discrete representations, absolute rules, learning as programming, and the hardware/software distinction (which places few constraints on what can be learned). These points are contrasted with connectionism's distributed representations, graded rules, learning by structural change, and software as hardware (the network's knowledge is defined by the structure of the network). To this list, Bates & Elman add that the non-linear dynamics of connectionism can make networks behave in unexpected ways, producing "truly novel outputs". Connectionism does seem to have all the right properties for capturing development, at least as a metaphor. Although the metaphor of production systems is attractive for modelling the stages *reached* in development (e.g., as accumulating rules) connectionist models provide insight to the *transitions* between stages.

The examples above have demonstrated that connectionism is useful in understanding development in a number of domains. It seems reasonable to suppose that a connectionist treatment could do the same for arithmetic.

4.3.2 Implementation and theory

VanLehn built Sierra as a variant form of a production system. On making this decision he comments (1990, p. 69):

> If one had to choose between production systems and connection systems as a representation language for knowledge about subtraction, then production systems seem much more plausible. Indeed it is not easy to see how a connection system could possibly generate the kind of extended, sequential problem-solving behaviour that characterizes students solving subtraction problems.

It is hard to disagree with VanLehn here. Much of this chapter has outlined the benefits of production system models for arithmetic, and they do indeed appear to be more plausible. However, at least one reason why production systems seem better suited is because there have been few connectionist models of anything as complicated as arithmetic. If it were easy to see how networks can generate "extended, sequential problem-solving behaviour", then perhaps VanLehn would not opt for production systems so quickly. This seems to be borne out by J. R. Anderson's comment that connectionist systems "…have been applied to such a small range of tasks that they are totally vague on the issue of control of cognition…in which the precision of productions systems is most exact" (1983, note 2, p. 307).

This section looks at the consequences of selecting a particular representation language, and concludes that a *theory* can be unduly restricted by the language. If the goal is to model a phenomenon with a virtual machine, then although symbolic AI has proved to be useful, there is no decisive reason to clinging to our current understanding of symbolic representation when designing the virtual machine. Section 4.3.3 demonstrates that there are other (connectionist) structures which can be considered as "symbols". These new structures and new styles of processing provide a different way to think about problems, and it is suggested that models of arithmetic will benefit from such ideas.

In selecting production systems, one is typically forced to build models that rely on the methods of symbolic AI. Rules, frames, plans, searching, and so on, have a role to play, but "the space of computational possibilities has hardly been entered" (Boden 1988, p. 260). This leaves the following open question: what kinds of computations are going to be useful in modelling the mind? VanLehn has placed his bet on symbolic AI, and this has no doubt influenced the development of his *theory* of arithmetic. But as Pylyshyn (1984, p. xvi) states:

> We cannot accept an operation as basic just because it is generally available on conventional computers. The operations built into production-model computers were chosen for reasons of economics, whereas the operations available to the mind are to be discovered empirically.

By modelling with an operation available on a conventional computer there is a risk of over- or under-estimating that operation's importance. In the worst case the model will incorporate highly improbable mechanisms, such as demanding a processor which is much faster than the human information processor. Another way of looking at this is to say that there is a danger of building "cognitive wheels". A cognitive wheel is:

> Any design proposal in cognitive theory…that is profoundly unbiological, however wizardly and elegant it is as a bit of technology.
>
> (Dennett 1984, p. 147)

As Clark (1985, 1986) notes, although the mind is treated as a black box system, we at least know that it is a "naturally occurring back box". Hence, a biological metaphor is suggested, in which it is insisted that cognitive science "be concerned with the development and testing of only such computational mechanisms as seem plausible in the light of whatever biological constraints may be expected to govern emerging natural structures" (Clark 1986, p. 47). Of course, these constraints are not detailed.

Logically there is no reason why nature could not have evolved a conventional computing architecture, in which case symbolic models would capture human performance exactly (Clark 1987a, 1989). In fact, some kind of higher-level "programming" language seems essential. As J. R. Anderson (1983, p. 3) notes:

> …it is totally implausible that we have evolved special facilities or "organs" for mathematics, chess, computer programming, or sculpture. People become expert at activities for which there was no possibility of anticipation in our evolutionary history…

However, given the kludgey way in which evolution constructs solutions (Clark 1987b), it seems unlikely that the symbolic architecture would be *just* that one which we use today.

Connectionism cannot claim to be empirically discovering the "operations available to the mind", and is it just as capable of producing cognitive wheels as symbolic AI. However it is clear that connectionism has changed the way we think about representational issues (discussed further in the next section). For example, given that connectionism is seen as being most applicable to the low-level processes, the following quotation suggests that it will also be essential in understanding the higher-level processes:

> Whatever the basic principles of language representation, they are not likely to be utterly unrelated to the way or ways that the nervous system generates visual representations or auditory representations, or represents spatial maps or motor planning.
>
> (Churchland & Sejnowski 1989, p. 42)

For "language representation" read any of your favourite higher-level functions. Although these issues are often dismissed as just implementational detail, they can have a profound effect on the models begin considered. If it turns out that some mechanism, X, is a very cheap computation, it could be used frequently. If X were expensive, then it would probably be used less often. So it seems that neither connectionism nor symbolic AI alone can expect to explain cognition: there is a sense in which all modelling involves forcing a phenomenon into the representation language you use. Some of the phenomena will fit naturally, and other aspects will not.

There seems no easy way to detect "unlikely mechanisms", and it is often only with hindsight that specific mechanisms may be declared as begin misguided. One possible example is any mechanism requiring centralized control. Connectionist research and neuroscience has suggested that control is distributed in the brain (Rumelhart & McClelland 1986b, pp. 134–135). A good candidate for a mechanism that is biased by a technology can be found in repair theory. In describing the impasse-repair process, Brown & VanLehn (1980) comment: "When a constraint or precondition gets violated the student, unlike a typical computer program, is not apt to just quit" (p. 381). What Brown & VanLehn have in mind is an analogy to the computer's error handling mechanism. This is made explicit by VanLehn (1990):

> When computers reach an impasse, they do not just turn themselves off…Instead they start executing an error-handling routine instead of the main program. Similarly when people are executing a procedure and reach an impasse, they…handle the impasse in some fashion (p. 41).

The idea is that certain errors are detectable in both human and machine, and once detected something can be done about the error. As discussed in section 4.2, this way of thinking has been successful in modelling arithmetic. However, the analogy breaks down for connectionism. The notion of an impasse is ill-defined for networks; networks will continue to produce output whenever they are given input. Thanks to properties such as similarity-based processing and automatic generalization, networks never "get stuck", unable to continue processing. This throws a whole new light on the impasse-repair mechanism: perhaps networks can automatically repair undefined situations just by virtue of having the right kind of architecture to start with—being a connectionist network, rather than a production system. All this depends, of course, on what networks actually do (chapter 5), and on the psychological importance of impasses (discussed in section 5.6.1).

The point to note here is simply that particular technologies suggest certain kinds of models and theories, and other technologies can force different interpretations. Despite his attempt to isolate his theory from his model, VanLehn concedes that at least one part of his argument "…relies on concepts and distinctions from traditional serial computer science…" (VanLehn 1990, p. 212). He adds: "…but those distinctions are rapidly changing as parallel computer science, especially connectionism, develops".

4.3.3 What is a symbol, anyway?

Even though an explanation of some cognitive skills may currently be best suited to the language to classical AI (i.e., operations on symbols), the nature of those symbols and operations are subject to revision. For example,

> …our ideas about what is *means* to 'operate on a symbol' are still heavily influenced by conventional AI implementations in which a symbol is a discrete internal state, manipulable (copyable, moveable) by a processor.
>
> (Clark 1987a, p. 12)

It is usually assumed that the symbols manipulated by production systems are the same kind of symbols manipulated by the mind. But there are other ways to look at symbols. An example is the reduced descriptions described by Pollack (1989b). Recursive auto-associate memory (RAAM) is based on Elman's simple recurrent network architecture (used in figure 4.4). Items in a structure (words in a sentence, perhaps) can be individually compressed into the hidden layer of the network (see figure 4.5). This compression includes the representation in the hidden layer from the previous time step, hence building up a fixed-width representation for the whole sequence. The network auto-associates the input to the output, thus ensuring that the elements in the sequence can be reconstructed from their compact representation. RAAM is not used to save memory by compaction; the usefulness lies in the representation's manipulative properties:

> They combine aspects of several disparate representations. Like feature-vectors they are fixed-width, similarity based, and their content is easily accessible. Like symbols, they combine only in syntactically well-formed ways. Like symbol-structures, they have constituency and compositionality. And, like pointers, they refer to larger symbol structures which can be efficiently retrieved. But, unlike feature-vectors, they compose. Unlike symbols,

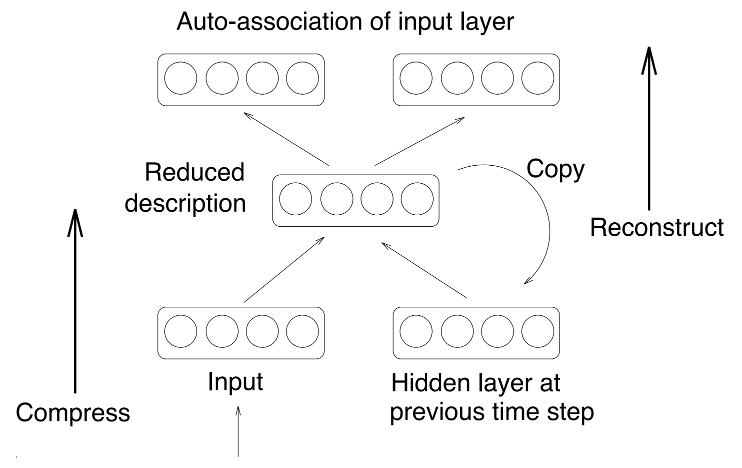
Auto-association of input layer
Reduced description
Copy
Reconstruct
Compress
Input
Hidden layer at previous time step

Figure 4.5. Recursive auto-associative memory (sequential version).

> they can be compared. Unlike symbol-structures, they are fixed in size. And, unlike pointers, they have content.
>
> Pollack (1989a, p. 529/530)

Pollack (1989b) has demonstrated that the RAAM has some form of generality, and is not simply memorising sequences. Further, these reduced descriptions have been used for inferencing. Pollack (1989a) describes training an associative network to transform reduced descriptions of structures like (LOVES X Y) into (LOVES Y X). That is, a simple network can embody the rule "if (LOVES X Y) then (LOVES Y X)". This kind of inferencing involves variable binding and sequential list chaining in classical AI. Yet for this network it is no more effort than an association, with an enhanced risk of being the wrong association. Chalmers (1990) reports using the same methods to transform active sentences into passive form, and argues that this form of computation ("holistic associative inferencing") is something that is new, and not available to classical AI.

This notion of representations having special properties is very similar to J. R. Anderson's (1983) notion of "salient properties." ACT* contains three distinct data types: temporal strings, spatial images and abstract propositions. Each data type has a set of salient properties associated with it. The salient properties of any data type determine which primitive operations are available. Temporal strings, for example, are similar to Lisp lists, but have a salient property of allowing order to be judged quickly: it is easier to say that March is before June, than it is to retrieve the months in between. In this way,

J. R. Anderson is attempting to follow Pylyshyn and empirically discover the operations available to the mind. It will be interesting to see if any of the salient properties suggested by J. R. Anderson are available ("for free") from connectionist representations.

The conclusion here is that connectionism is changing the way we think about representation. It has some surprising properties, and it may be the case that connectionism can produce some surprises in other domains (i.e., arithmetic). It is not just connectionism that can revise representation; ideas are also flowing in from areas such as genetic algorithms (Koza 1992) and dynamic systems (van Gelder 1992).

4.3.4 *Comments*

Connectionist networks exhibit many mind-like properties. Although certain cognitive functions may best be explained at the level associated with classical AI, there seems to be a case for investigating these processes from a connectionist point of view. As J. R. Anderson (1983, p. 41) notes: "Using a computer analogy, the cognitive system is like a rich and complex programming language with many special hardwired features." The ideas about representation and styles of processing being developed by connectionism may well help identify the "special hardwired features." If hybrid systems are developed in which parts of a production systems are implemented in connectionist technology (Touretzky & Hinton 1988; Stark 1992) there are likely to be aspects of the representation language (the salient features) which could be exploited by the production system.

It seems that connectionism may be well suited to capturing developmental phenomena in general. For arithmetic in particular, it could be that some of the assumptions made by VanLehn (1990) and Young & O'Shea (1981) may be changed when viewed from a connectionist perspective. This is not to suggest that connectionism necessarily offers a "better" account of cognition, or that production system models can be ignored. Rather, there is some promise in looking at arithmetic from a different—in this case, connectionist—point of view. The "true" story is much more likely to be hybrid than purely connectionist or purely symbolic.

Having established these points, and having looked at the kinds of bugs and models that surround children's arithmetic, a connectionist model can now be described.

CHAPTER 5

A Connectionist Model of Multicolumn Arithmetic

Before detailing the connectionist network used to model arithmetic, this chapter first outlines the known features of arithmetic that constrain the model. The architecture, input and output representations and training environment are then described. The results of simulations—the errors the network makes when solving multicolumn problems—are presented, and the network's behaviour is analysed. Finally there is a discussion of how the network relates to Sierra, and how notions like bug migration and impasses can be accommodated.

5.1 Constraints on the model

Few people do long multiplication "in the head", preferring instead to use an external representation on paper. This kind of mundane observation needs to be incorporated into the design of any model in order to constrain it. These constraints are listed in this section.

One thing which is *not* constrained is the grain size of the model, yet this is often what makes or breaks a model. Choosing the wrong set of operators can mean that the model apparently fails in its task. An example of this is VanLehn's decision to include "leftmost column" as part of the problem representation; without it Sierra would not be able to account for the bug always-borrows-left (figure 4.3). Of course, empirical validity is not the only measure of a model (as discussed on page 103), but without a reasonable fit to the data it becomes difficult to judge the explanatory power of a model. These issues are taken up throughout this chapter.

5.1.1 *What we know about multicolumn arithmetic*

Taking assumptions and observations from VanLehn (1990) and Young & O'Shea (1981), it is possible to present a list of some of the things that are known about arithmetic:

Children don't understand it. This point was discussed at length in section 4.2.3, concluding that multicolumn arithmetic is best described in terms of a procedural (syntax-only) set of skills.

It takes time. Many connectionist models solve whole problems in a single step: input is presented, and an output is computed. Multicolumn arithmetic, however, is characterized by its sequential nature. A network in this domain must be able to produce a series of actions even when the external input does not change.

Problems vary in length. There is no point in building a network which can only solve problems containing a limited number of digits. The power of multicolumn arithmetic stems from the fact that a simple set of arithmetic procedures can be applied to problems of unlimited length. This point and the previous one are the main reasons for selecting a *recurrent* network as the architecture of the model.

There are eye movements. Suppes et al. (1983) measured the eye movements made by subjects when solving multicolumn arithmetic problems. It was showed that subjects tend to follow the normative algorithms taught in schools. That is, for addition the eye focusses at the top of the page, scans down the column, then jumps to the start of the next column, and so on. This information was used by Suppes et al. to build a model of eye movements (discussed later). Eye movements are not accounted for in the symbolic models of buggy arithmetic. In the Young & O'Shea account the operation `ReadMandS` deposits the digits of the current column into the working memory; the operations `ShiftLeft` moves attention to the next column. Hence eye movements are below the level of detail addressed by the current symbolic accounts.

You don't always mark the page. The subject is not marking the page when focussing on a digit, or recalling the product of a multiplication. Again this can be viewed as a question of grain size: It is possible to build a model in which each operation is quite complex, resulting in a mark being made on the page, although no existing models suggest such a coarse level of detail. Given the serial scanning nature of arithmetic, the model described in this chapter includes a simple form of eye movement, whereby a focus of attention is shifted digit by digit over the problem. Each step does not necessarily mark the page, which means that the network will have to rely on some kind of internal memory to know what to do when the input does not change.

Arithmetic is learned in steps. Many connectionist networks are trained by presenting the system with a broad set of instances from a target mapping. As described in section 4.2.3 there is a definite curriculum for arithmetic. This means the training set for the model will be "staged". First the network will be trained on simple addition problems involving just two digits with no carrying. Then, two column problems are introduced and trained along with the previously learned problems (to avoid catastrophic interference; see McCloskey & Cohen 1989; Ratcliff 1990; McRae & Hetherington 1993). It is not the case that staged learning makes this problem easier to learn (as it did for Elman 1991). Rather, the network described below will learn the training set three or four times faster when the problems are presented in a single training set. The cost of this speed-up is that the bugs exhibited by such network include such unlikely actions as carrying on simple problems like 1+1. It could be argued that, early in learning, children solving 1+1 would not have the concept of carrying, hence such errors are never going to be observed. So to avoid such star bugs, and to mimic the training environment of children, the input to the network is staged.

Learning is by example, not verbal recipes. This was discussed in section 4.2.3. This is convenient for a connectionist model because it means the system can be trained from actual example problems, rather than trying to get a network to (somehow) interpret an encoding of the procedures for arithmetic.

There are bugs and slips. Run time slips, such as $3\times4=8$, were considered in part I. Here the focus is on procedural misconceptions, where mistakes are not due to "faulty recall", but result from mislearning. To isolate the procedural aspects of arithmetic, much of processing involved in solving a problem is done outside the network. External activities include keeping track of the current focus of attention, computing products, and keeping running totals. Training a network to output the right steps in a procedure ("multiply now", "remember this number") is hard enough without all the additional details of register storage, and remembering the multiplication facts. The result is a connectionist finite state machine. Embedding the facts network from chapter 3 inside this state machine would be an interesting project, but it is not taken up here.

Bugs occur during testing. Many bugs show themselves when the subject attempts to solve previously unseen problems (see section 4.2.3). After training a network on a set of problems, any over-generalization can be assessed by presenting a test set of slightly harder problems—namely the next set of problems in the curriculum sequence. In some circumstances it may be possible to find a training set that leads to near-perfect generalization. With this in mind VanLehn (1991b) discusses the notion of a pseudo-student—a program that will take a textbook lesson sequence and return a list of the misconceptions students are likely to acquire from that particular set of problems. Using this knowledge it will be possible to adjust the textbook's lessons to reduce the likelihood of students acquiring misconceptions.

Many of the assumptions of the model are based the around assumptions of VanLehn's theory. Others may disagree with these assumptions, for example about learning being from examples only. However, to give this domain a connectionist treatment, it seems sensible to start from the assumptions of an established theory, rather than attempt to reconstruct a new one.

5.1.2 What we don't know about multicolumn arithmetic

The above points help to narrow down the design of a model, but there are a number of fundamental questions which throw the design space wide open. Three particular unknowns are:

What is read. Although the eyes move this does not mean we know what is actually read from the page. Do subjects just look at single digits, or is the context of the digit, such as the space around it, important? Reading arithmetic problems is nothing like reading text: Suppes (1990) found that two, quite different models were needed to capture eye movements in these two domains.

What is done. In terms of virtual machines, what are the operations carried out on an arithmetic problem? This is not just a problem of getting the right grain size, but of getting an appropriate set of operations, as discussed at the start of this section. Given that there are many instruction sets that can perform a given task, what are the criteria for favouring one over another? With enough degrees of freedom it is always possible to produce a fit to the data, so empirical adequacy is only part of the story.

How we navigate problems. What kind of cues does a subject use when solving a problem? When looking for the top of a column, does the eye scan up until it finds a large enough area of space? Or is there some kind of peripheral information allowing a fast, ballistic movement?

These unknowns are mostly ignored in all the accounts of arithmetic malrules to date. What is missing is a solid understanding of the perceptual processes underlying our symbolic arithmetic abilities. As Suppes et al. (1983) comment: "Without such a [perceptual] component we have no way of representing the mental operations a person actually uses to process the written symbols presented to him" (p. 342). Given the lack of empirical research in this area, I have used many of the detail from the work of Suppes et al. to design the connectionist model.

The Suppes et al. model of eye movements

The register machine presented by Suppes et al. (1983) was built to capture eye-fixation durations and saccade directions for multicolumn addition and subtraction. In particular the aim was to ensure that eye movements could be correlated to steps in the register machine. The model's operations included:

- A stimulus supported register (SS) which contains the element (digit, rule mark, space) currently being looked at. The SS register is subject to decay when the focus of attention is moved.
- A nonstimulus-supported register (NSS), which is not subject to decay. This register is used to accumulate an answer.

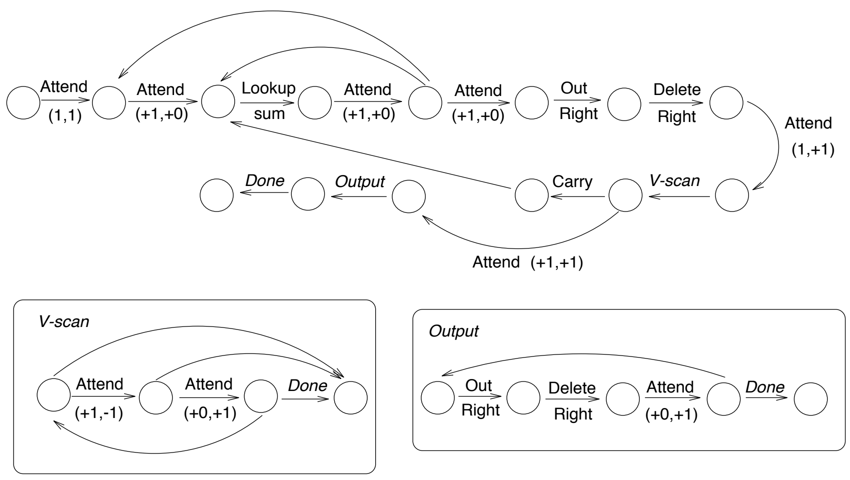

Figure 5.1. Transition network for long addition based on routines by Suppes et al. (1983). Digits in the problem are represented in a coordinate system (row,column), relative to a top-right origin (1,1). A movement is denoted (± row,± column). Subprocedures are used for vertically scanning a column.

- A copy operation allows the contents of SS to be moved to NSS.
- The left- or right-most digit of a register can be written out. For two digit numbers this obviously means having access to the tens and units of the number.
- Arithmetic facts are supplied by a "lookup" operation.
- Conditional branch points are used—"gotos" which are dependent upon the contents of a register.
- The problem is represented as cells in a grid.
- The focus of attention is directed by an "attend" operation, which is either absolute (attend to grid position a, b), or relative (shift attention by $\pm a$, $\pm b$).

Using these elements, Suppes et al. found that a register model could be built in which the operations correlated with eye activity (figure 5.1). That is, eyes tend to move when the register machine requires movements (attend), and stay put during other cycles (copy, lookup, etc.). Of course, eye movements are highly stochastic, and the measures of correlation that Suppes et al. used are necessarily complex. Despite the success of

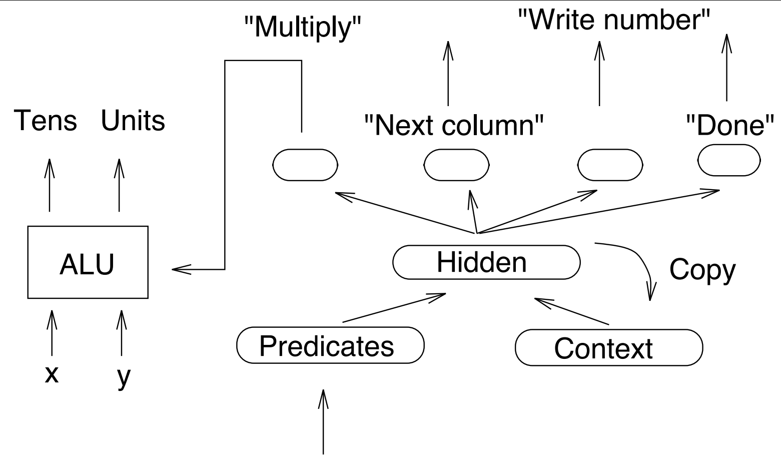

Figure 5.2. Architecture of the model. Outputs from the network drive external machinery (ALU) to perform tasks such as addition and multiplication.

the model there are many more aspects to consider—e.g., the use of the grid system of coordinates, and eye movement velocities.

The basic elements from this model of arithmetic eye movements were used as the starting point for the connectionist model described in the next section. Although Suppes et al. did not attempt to model arithmetic malrules, their analysis of of the perceptual systems should be an important component of any arithmetic model.

5.2 Architecture of the model

Having looked at the origins of the ideas behind the connectionist model, it is now possible to specify the details. The architecture is shown in figure 5.2. An encoding of the problem is presented to a recurrent network, which activates a set of 35 hidden units, which in turn activates an output layer representing actions to be carried out. These actions drive external mechanisms to update the problem in various ways, such as writing down a number, moving the focus of attention, or computing a sum or product. To enable the network to process sequences, the hidden layer is copied to the context layer at each time step acting as a memory of the previous state.

We can view the problem of learning arithmetic algorithms as learning to be a finite state machine (FSM). VanLehn (1990) also found it useful to describe long subtraction in

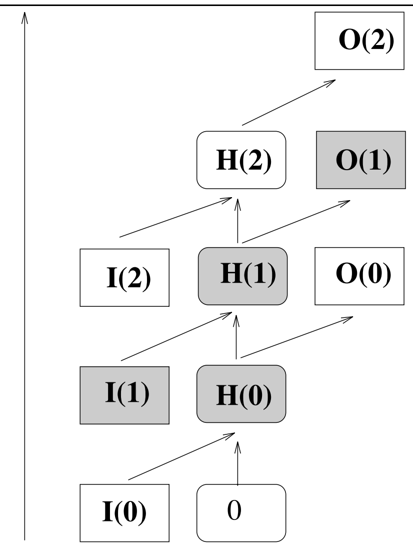

Figure 5.3. Backpropagation through time (BPTT). The shaded area shows a single network. I(t) is the input at a particular time step, t. H represents a hidden layer, and O represents an output layer. The training of a recurrent net is reduced to training a many-layered feed-forward network.

terms of transition networks.

There are plenty of connectionist models that can learn to be FSMs (Rumelhart, Hinton & Williams 1986; Elman 1988; Servan-Schreiber, Cleeremans & McClelland 1988; Cottrell & Tsung 1989; Allen 1990; Williams & Zipser 1989; Fahlman 1991). I chose to use back-propagation through time (BPTT), as proposed by Rumelhart, Hinton & Williams (1986, pp. 354–361). BPTT involves stacking-up a copy of a feedforward network for each step in a sequence, and then "rewinding" at the end of the sequence, backpropagating error to the very first pattern in the sequence. Although it seems to be an unpopular algorithm, it has distinct advantages over other candidates: it can learn sequences that simple recurrent networks find difficult (Maskara & Noetzel 1992); it shows better performance, in terms of generalization and learning success, that the real-time recurrent learning algorithm (Zipser 1990); and backpropagation is better understood than more recent constructive algorithms, such as cascade correlation.

BPTT works by making one copy of the network for each step in the problem. As the context layer is the hidden layer from the previous time step, it is possible to re-draw a BPTT network as in figure 5.3. At each step of the forward pass, the activations of all the layers are recorded. After the last output has been produced, error is backpropagated down the virtual network in the usual way. In the figure, error is propagated from O(2) to H(2), allowing error information to be computed from H(2) to H(1). Also, at this point O(1) contributes the error associated with the output at time step 1. This continues with H(1) backpropagating error to H(0), and so on. Note that the weights from I(1) and H(0) to H(1) and O(0), are exactly the same as the weights from I(2) and H(1) to H(2) and O(1). After the error has been backpropagated to the start of the network, the weights are updated using the generalized delta rule (Rumelhart, Hinton & Williams 1986).

The BPTT algorithm differs from the use of backpropagation in simple recurrent networks (SRNs) because error is preserved across the hidden layers. SRNs (as described by Elman 1988) have the feed back from the hidden layer to the context layer, but do not stack up copies of the network. Error is computed after each pattern. When using a near minimal number of hidden units for a task, SRNs can carry information about certain contingencies over a number of time step. That is, SRNs can use knowledge from previous time steps to influence actions much later in the sequence. "Such information is maintained with relative ease if it is relevant at each time step; it tends to get lost when intervening elements do not depend on it" (Cleeremans, Servan-Schreiber & McClelland 1988, p. 372). BPTT obviously does not have this problem as the whole time sequence is drawn out. Preliminary tests identified that the problem of multicolumn arithmetic, as presented here, requires BPTT. However, SRNs can learn this task, but only by using many more hidden units than required (Maskara & Noetzel 1992).

Cleeremans et al. (1988) found that recurrent networks of this kind, when trained on example strings from a finite state grammar, can become perfect recognizers for that grammar (see also Servan-Schreiber et al. 1988; Servan-Schreiber, Cleeremans & McClelland 1991; Allen 1988; Elman 1988). A SRN was presented with strings from an artificial grammar (figure 5.4). At each time step one letter was presented, and the network's task was to predict the next letter in the sequence. Given that any node in the grammar may have a number of successors, it is not possible to correctly predict the next letter. A particular letter's successor will depend on the context of what other letters have been processed—the choice of successor will depend on which node has been

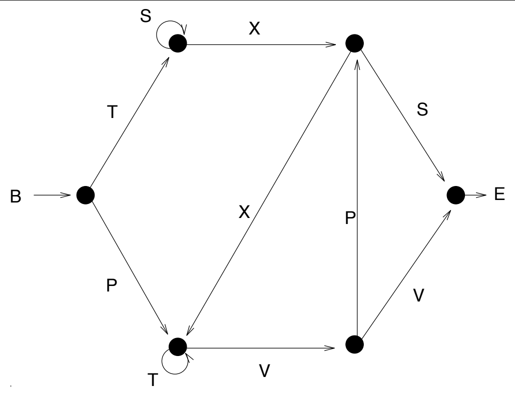

Figure 5.4. The grammar used by Cleeremans et al. (1988, p. 374). B is the designated start symbol, and E is the end symbol. Traversing an arc generates a symbol (B,P,S,T,V,X, or E). An example string is: BTSSSXXVVE

reached in the grammar. The network was trained on 60 000 randomly selected strings from the grammar. When testing the network any output unit with activation above 0.3 was taken as a prediction of a legal successor. When the actual successor was not one of the predicted successors, the network was said to have rejected the string. The string is accepted if only good predictions have been made when a designated terminal symbol was reached. In a test of 70 000 randomly generated strings, 0.3 per cent happened to be grammatical. The network accepted all the grammatical strings, and rejected the others. When the hidden unit activations were analysed, by clustering the hidden vectors according to their similarity, it was found that the vectors grouped according to nodes in the grammar. Given that multicolumn arithmetic can be viewed as learning to be a FSM, it seems that this kind of recurrent network is appropriate for the task.

5.2.1 Input and output representations

Having selected an architecture that is powerful enough to learn arithmetic procedures, the next step is to specify the input and output representations. Figure 5.5 shows a general framework for representing long multiplication using a grid. To solve a problem represented in this way requires three types of operations:

- Attending to particular areas of the problem to read and write digits. To do this the network moves a "focus of attention" around the grid.

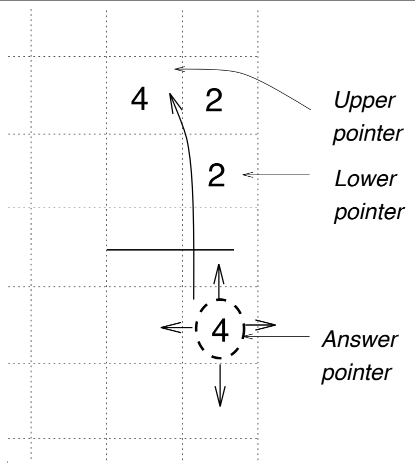

Figure 5.5. Problem representation

- Computing arithmetic facts, or other useful predicates. The lookup of arithmetic facts is done externally to the network, and a small number of truth values (e.g., "in leftmost column?") are presented as input.
- Controlling the attention of the network, knowing when to write down a digit, etc. The training set and choice of output operations determines the control strategy.

Three external registers are used to keep track of the network's progress through the problem. One register is used to point to the current digit being processed in the upper-row and another is used to point to the digit in the second row of a problem. These registers are not necessary for addition, but are for multiplication. The third register is used to locate the place where the answer should be written. These registers are initialized to default positions at the start of a problem, and it is the network's responsibility to update them by turning on appropriate output units associated with increment operations. When the network needs to read the upper-row digit, the "jump to the top row" output unit will be switched on.

There are may other ways that action could be represented without using external registers. For example, the output layer could be split into an operation-argument pair. In this case moving the focus of attention would mean switching on the "attend" operator in the first set of outputs, and switching on some encoding of position in the second set of units. This style of representation seems more likely to succumb to "memory slips"

Movement	Registers
`top_next_column (TNC)`	`store_mark (STR)`
`jump_answer_space (JAS)`	`zero_accumulator (ZAC)`
`jump_top_row (JTR)`	`next_answer_row (NAR)`
`left (LFT)`	`next_bottom_column (NBC)`
`right (RHT)`	`inc_answer_column (IAC)`
`up (UP_)`	`inc_top_column (ITC)`
`down (DWN)`	`add_start_position (SAD)`
`read_carry (RDC)`	`start_multiplication (SMU)`

Writing	Others
`write_units (UNI)`	`add_mark_to_accumulator (ADD)`
`write_tens (TEN)`	`compute_product (MUL)`
`mark_zero (MKZ)`	`draw_rule (RUL)`
`mark_carry (MKC)`	`done (DON)`

Table 5.1. Actions that the network can perform (and abbreviations used in some figures).

by getting the position encoding wrong. Making use of external registers focusses the model on procedural aspects of the problem—knowing that focus must be moved to the top row, but without having to store actual coordinates. Positioning errors can occur, though, if the network does not increment or reset the counters at the appropriate time.

Twenty-four output actions can be performed (table 5.1). For each operation there is an associated output unit in the network (a 1-of-24 encoding). This set of operations is adequate for multiplication and addition, and was used in all the experiments described below. Variations in the operations, and the effects this has on the model, is a topic for future investigation.

The top-row, bottom-row and answer registers have increment operations and reset operations. There is also the notion of a "current focus": after the network has focused on a digit in the top row, for example, it can move relative to that position (up, down, left or right), or read the carry mark if there is one in that cell. All these registers are reset to appropriate starting positions when the network activates either the `add_start_position` or `start_multiplication` units at the start of a problem.

As the focus of attention moves around the problem, the contents of the current cell is read (cf. Suppes et al.'s SS register). The contents may be stored in another register

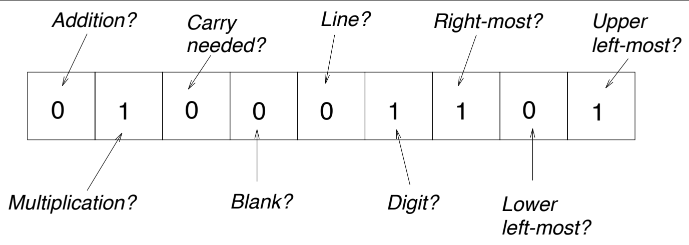

Figure 5.6. Input representation.

(NSS) and used in computations. When addition or multiplication is called for, the calculation uses the current digit in focus and the last stored digit, and places the result in an accumulator—or adds the result to the accumulator in the case of addition. The accumulator's contents is accessed, mostly to write down a digit, by the tens column or the units field. A special operation (`mark_carry`) writes the tens part of the accumulator at an appropriate place relative to the current focus of attention.

Two operations are particular to multiplication. The `mark_zero` operations simply writes a zero at the current focus of attention. This is useful for annexing zeros into the partial product (see figure 4.1 on page 89). After completing the partial product, but before the addition begins, it is customary to draw a line under the partial product. The associated operation, `draw_rule`, does this.

A network has solved a problem when the `done` output unit is active. At all times the operation carried out is the one associated with the most active output unit.

A number of output actions involve "jumping" the focus of attention to certain points in the problem, such as the top of the column. There are other possibilities: for example, rather than allowing a `top_next_column` action, there could be a loop of actions moving the focus of attention up the page until it reached the top of the column. This would result in less output actions, but much longer training sequences. Indeed, it is possible to navigate around a multiplication column with only one register, an accumulator, and the ability to move up, down, left or right. But not only would the sequences be very long, this model also runs against introspective accounts of how one solves a multiplication problem. Exactly how people navigate maths problems is an interesting and open topic which requires further empirical research.

The input to the network is a vector of 7 bits encoding information about the cell at the current point of focus, and 2 bits to indicate the task (figure 5.6). The task bits are set at the start of processing and remain unchanged for the duration of the problem. If the task is multiplication the first bit is set, otherwise the second bit is set. As the network is not concerned with the actual numbers in any given problem (remember the calculations are done externally), the input only has to indicate if the current cell contains a digit, a rule line or a space. Three bits encode this information. That is, if there is a digit in the current cell, the sixth input bit is set—there is no encoding of which digit it is. A further three bits are set when the focus moves to a column that is the rightmost column, or the leftmost as defined by the second row of the problem, or leftmost as defined by the first row of the problem. These flags correspond to important points in the problem, such as when to stop processing, or when to move on to the next multiplier.

The remaining input bit is set when the accumulator exceeds 9 (i.e., when a carry is needed).

To summarize, the input to the network is an encoding of the current cell of the problem. The previous hidden layer output, initially all zeros, are also presented as part of the input. This allows the hidden layer to act as the state of a finite state machine: next state $= f$(previous state, external input). The output of the network is an instruction to move the focus of attention, or perform a calculation or update one of the external registers.

5.2.2 Training

Learning an algorithm for long multiplication involves learning sequences made up of the primitives mentioned in the previous section. For example, the problem…

$$\begin{array}{r} 2 \\ \times\ 3 \\ \hline \end{array}$$

…is represented with the training sequence shown in table 5.2. Seven steps are required to solve a two digit multiplication which does not involve carrying. Other sequences are longer: 12×59 is solved with 64 steps.

Task	Cell	Output	Comments
×		`start_multiplication`	Look at '3'
×	Number	`store_mark`	Remember the '3'
×	Number	`jump_top_row`	Move to the '2'
×	Number	`compute_product`	Multiply digits
×	Number	`jump_answer`	Move down
×	Space	`write_units`	Write '6'
×	Space	`done`	

Table 5.2. An example of a training sequence for 2×3. The last three input bits, information about leftmost and rightmost columns, are not shown here.

As the input representation registers information about the presence or absence of a digit and the state of the accumulator, but *not the actual digits in the problem*, the training set need only contain one instance from each kind of arithmetic problem. That is, rather than training the network on 1×1, 1×2, 1×3, 2×3, 9×1, and so on, the training set only contains one problem for "one column problems without carrying." This is because 9×1, 1×1 and 2×3 all produce exactly the same sequence of output vectors, or follow the same path through a finite state machine.

For the additions 0 + 0 to 999 + 999 there are 40 different sequences that need to be learned; for 0×0 to 999×999 there are 362 sequences. A subset of 27 problems (10 addition, 17 multiplication) were trained on. The problems were arranged in a curriculum of increasing difficulty (table 5.3). This problem set was not optimized, and no other combinations were tried. The problems were selected to conform with the difficulty sequence found in a school textbook (Howell et al. 1979), and the sequence identified by Cox (1974, see discussion in appendix A).

One training set was created for each of the problems. The first set just contained the 9 input-output pairs for 1+1. The second contained the 9 for 1+1, plus the 11 steps for 1+1+1. In this way, each training set included the previous set. The last training set contained a total of 1156 training pairs.

The network was trained with a low learning rate (0.01) and no momentum using backpropagation through time. These parameters were determined by running a number of networks with various learning rates, momentum rates, tolerances (see below), and different numbers of hidden units. I selecting the combination which learned reliably

1: 1 + 1	8: 101 + 109	15: 12×5	22: 12×50
2: 1 + 1 + 1	9: 101 + 99	16: 12×9	23: 12×55
3: 11 + 11	10: 101 + 899	17: 1×11	24: 12×59
4: 11 + 1	11: 1×1	18: 11×11	25: 12×90
5: 1 + 9	12: 2×5	19: 1×111	26: 12×95
6: 1 + 19	13: 11×1	20: 12×15	27: 12×99
7: 100 + 100	14: 111×1	21: 12×19	28: 111×11

Table 5.3. Problems used to train the network. The 28th problem was used for testing the 27th network and was not trained on.

(with no local minima) and quickly.

It was found that rather than using 1.0 and 0.0 as target values, learning times were reduced by using 0.9 and 0.1. This helps by ensuring that the weights do not push the output activities too far along the sigmoid function, so that the derivative of the activation function is larger than it would be if the network was trained to 1.0 and 0.0. It is similar to the sigmoid prime offset used in chapter 3. Sigmoid prime offset could not be used with this recurrent network because it made the weights "explode"—suddenly reach very large positive or negative values. The reasons why this happens remains unclear.

A performance measure was used to determine the point at which a network could solve the problems in the training set. An output vector was classified as "correct" if each element of the vector was within a tolerance of the target value. The tolerance was set at 0.2, meaning that an individual output unit was "correct" if it was ± 0.2 from the desired value. Note that this measure was not used in the computation of error with backpropagation—it just provides a way to monitor the training progress which is more intuitive that total sum squared error. Performance is the number of correct output vectors as a percentage of the number of patterns in the training set.

Training on a particular set of problems continued until 100 per cent correct performace was achieved or 10 000 epochs had passed. Then the next training set was introduced, and learning continued. In most cases the performance measure stopped the training. In the cases where 10 000 epochs were reached, it was found that the networks could solve the problems correctly. A less than 100 per cent correct network can still produce the correct output sequence because during testing the most active output unit is taken to be the output of the network.

The training was also “teacher forced”: the network always received the input that would be expected if it had produced the correct sequence of actions, regardless of the sequence actually produced. During testing, however, the output from the network is acted on, not the desired output. Likewise, the actual input that results from these changes is presented on the next time step, not the expected input.

Given that the training set changes in size, it is easier to report learning times in terms of the number of input-output pairs processed than in terms of epochs (number of passes through the training set). One input/output pair constitutes a forward and backward pass through the network. It took 45 756 input-output patterns to learn 1+1 (5084 epochs), a further 660 pairs to learn 1+1+1, and a further 265 430 to learn 11+11. The learning times increase when introducing new concepts (see figure 5.7). A total of 3 857 965 patterns were needed to learn addition, and 33 271 488 patterns had been processed after the 27th sequence had been learned.

5.3 *Results*

The training of a network is not of central importance here. The aim is to see how the network performs on unseen tasks, and what kinds of mistakes it makes. In this section results are presented from testing trained networks on multicolumn arithmetic problems. The testing procedure begins when a problem is presented to the network and activation propagates through the network. The most active output unit is taken to be the network’s response. The operation associated with the response if executed—there is no teacher forcing. This continues until the output is “done”. As demonstrated in the previous section, learning is not problematic. After training on a particular problem set, the network correctly solves the problem in the set. Hence testing is only on problems the network has not encountered before.

First note that there is some generalization. For example, after training on 1+1, the network can solve 1+1+1 (the next problem in the curriculum), and 1+1+1+1, but no further. After the third training set has been learned the network can add a row of unlimited length, tested on a row of 200 digits, with a sum less than ten. This limited

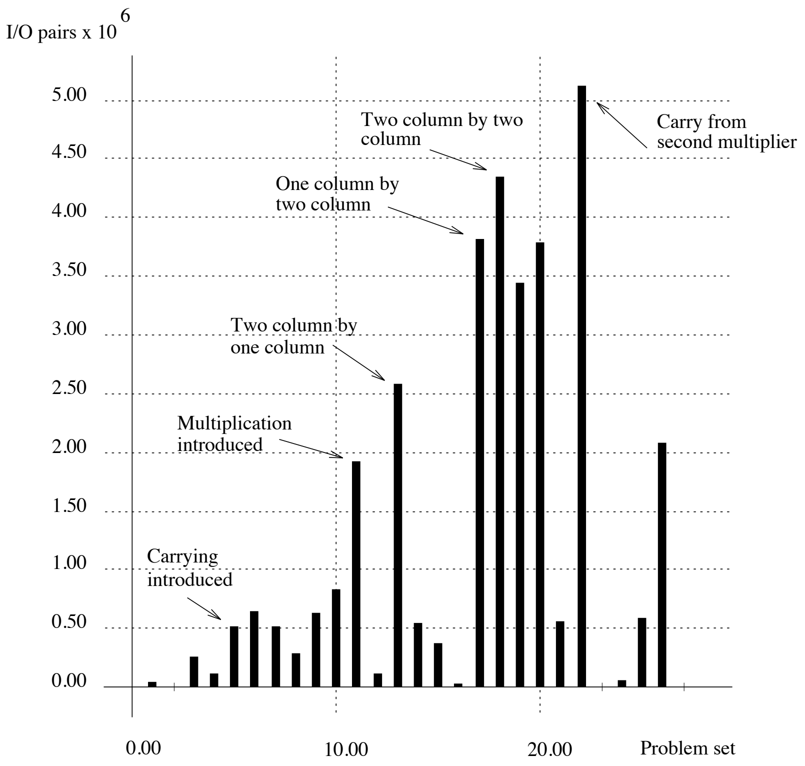

Figure 5.7. The learning time required for each training set. Each bar represents the number of input-output pairs needed to learn a particular training set in addition to the time required for previous training sets. The taller the bar, the more difficult the problem. The problem set numbers on the x-axis refer to the problems in table 5.3.

generalization and other generalizations will be discussed in more detail later.

On other problems the network outputs the wrong operation, leading to various kinds of mistakes. Some of these mistakes are interesting and bug-like. For example, solving 59×12 involves solving $118 + 590$. Although the network can correctly solve the addition part alone, when presented as part of the multiplication one action (`mark_carry`) was skipped, resulting in:

$$\begin{array}{r} 5\ 9 \\ \times 1\ 2 \\ \hline 1\ 1_1 8 \\ +5\ 9\ 0 \\ \hline 6\ 0\ 8 \end{array}$$

This is a surprising result from embedding addition within multiplication, and is explaned in section 5.4. Another interesting and plausible bug comes from testing the network on a problem it has never seen before:

$$\begin{array}{r} 1\,1\,1 \\ \times \quad 1\,1 \\ \hline 1\,1 \\ + \qquad 1 \\ \hline 1\,2 \end{array}$$

In the following example the network correctly multiplies by the first multiplier and then quits, ignoring the second multiplier. This is the bug quits-after-first-multiplier:

$$\begin{array}{r} 1\;\;2 \\ \times 1\;\;5 \\ \hline 6_1\,0 \end{array}$$

The same network behaves differently in other situations:

$$\begin{array}{r} 1\;\;2 \\ \times 5\;\;3 \\ \hline 8\;\;6 \end{array}$$

Here the network correctly executes the first multiplications (3×2=6, 3×1=3), but then adds the 5 to the accumulator (3+5) to get 8. This is an example of the way the network can mix parts of addition with multiplication.

Of course many of the bugs are highly implausible (star bugs). For example, one network was set the task 12×90, and wrote all over the page, resulting in:

$$\begin{array}{ccccc} & & 0 & 1 & 2 \\ & \times & 0 & 9 & 0 \\ \hline & & & 0 & 0 \\ + & 0_1 & 0_1 & 8 & 0 \\ \hline & 0 & & & \end{array}$$

In order to asses how well the model accounts for bugs, the following experiment was carried out. The weights of the network were saved after every training set, resulting in 27 weight matrices. Then each of the 27 networks was tested by presenting it with the problem in the next training set—a problem the network was not trained on. The

Bug type	+1	+2	+3
Correct	24.00	16.67	14.29
Observed	36.00	35.42	37.14
Unobserved	40.00	47.92	48.57
Totals	100.00	100.00	100.00
Star	32.00	31.25	32.86
Plausible	8.00	10.42	7.14
Combination	0.00	6.25	8.57

Table 5.4. Classification of the behaviour of the model when tested on unseen problems. Column 1 percentages are from 25 problems, column 2 from 48 problems, and column 3 from 70 problems. The unobserved bugs are broken down into star, plausible and combination bugs.

behaviour of the network was classified, by hand, as being correct, an observed bug, or an unobserved bug. An unobserved bug is one that has not been recorded for human subjects, and therefore does not appear in the appendicies. The first column of the top half of table 5.4 shows the results as a percentage of all the problems solved.

This process was continued by testing each network on the problems in the next two training sets, and in the next three training sets (columns 2 and 3 of the table).

As can be seen, the model predicts far too many unobserved bugs: around 50–60 per cent of all behaviour is an observed bug or correct, but at the cost of having to generate over 40–50 per cent unobserved bugs. The unobserved bugs can be broken down into: star bugs, which I believe will never be observed in a study of human bugs; plausible bugs, which do not appear in the appendices, but I believe that they might be observed; and combination bugs, which can be described by a combination of two bugs, of which at least one is unobserved but plausible. Examples of these categories are presented below, taken from all the problems solved by the networks in the experiment (the last column of table 5.4).

Correct behaviour

The networks generalized correctly on a total of 10 problems. Examples include: the network trained on 1+1 correctly solved 1+1+1; after learning 11+1, 100+100 was solved; 1×1 generalized to 2×5; once 12×59 was learned, the network correctly solved 12×90.

Star bugs

A total of 23 star bugs were found, indicating that there needs to be some additional constraints on the model. Four of the 23 bugs result from allowing the network to write anywhere on the page, including the initial problem, and rule-marks. Endless looping accounts for 5 of 23 bugs. In 3 cases the answer was not recorded—a serious star bug, as subjects usually write something.

The majority of star bugs (11) were due to the network repeating a large operation, such as repeating the addition sequence after adding a partial product. This often involves writing over marks that are already on the page.

Plausible bugs

There were three bugs that seemed plausible. The first occured three times. The network fails to mark the carry from the tens column, but does for the ones column:

$$\begin{array}{r} 1\ 0\ 1 \\ +\ \ 8\ 9\ 9 \\ \hline 9\ 0_1\,0 \end{array}$$

There was a kind of "operation confusion" (2 occurrences), where the first multiplier is used correctly, but the result of the second multiplication is added to the second multiplier. An example of this was given on page 135 (12×53).

There was 1 occurrence of a bug in which an answer is repeated, but skipping over the digits in the last column. In the example, 2×4=8, then the network scans down the second column, ignoring the numbers, and writing the previous product (8) in the answer.

$$\begin{array}{r} 1\ 2 \\ \times\ \ 3\ 4 \\ \hline 8\ 8 \end{array}$$

Plausible combinations

I classified four behaviours as combinations of bugs of which at least one was unobserved, but plausible. There were 2 occurances of a combination of does-not-record-100s (only observed for multiplication), and does-not-add-carry. Both cases were on addition problems:

$$\begin{array}{rrrr} & 1 & 0 & 1 \\ + & & 9 & 9 \\ \hline & & 9_1 & 0 \end{array}$$

The next behaviour combines does-not-record-100s with a bug in which the carry is ignored in the tens column which is similar to the observed bug does-not-carry-ones. Again, this is an addition bug (2 occurrences):

$$\begin{array}{rrrr} & 1 & 0 & 1 \\ + & & 9 & 9 \\ \hline & & 0_1 & 0 \end{array}$$

The first multiplication combination occured once, and involves the bugs does-not-add-partial-product and last-multiplication-skipped:

$$\begin{array}{rrr} & 1 & 2 \\ \times & 6 & 1 \\ \hline & 1 & 2 \\ 1 & 2 & 0 \end{array}$$

Note that the *classification* here is of a combination of bugs, but the behaviour derives from one mistake. In the previous example, the processing stopped after the 12 from 2×6=12 was written. No attempt was made to add the partial product. It is not clear how further training on adding partial products would change this behaviour. For children, remediation on an individual bug in a bug combination will presumably remove the bugs independently. I have found no evidence in the literature for this one way or the other. Nevertheless, future work should explore the possibility that apparent bug combinations in the network can be removed individually.

The final combination again involves does-not-add-partial-product, and also repeats part of the multiplication (1 occurrence):

$$\begin{array}{rrr} & 1 & 2 \\ \times & 6 & 1 \\ \hline & 1 & 2 \\ 1 & 2 & 0 \\ 1 & 2 & 0 \end{array}$$

Bug name	Rank	+1	+2	+3
ignores-10s-column	10	22.22	23.53	19.23
does-not-carry-ones	1	0.00	5.88	11.54
does-not-record-100s	16	22.22	11.76	7.69
does-not-raise-carry	9	11.11	5.88	3.85
quits-after-first-multiplication	n/a	22.22	29.41	30.77
quits-after-first-multiplier	7	22.22	23.53	26.92

Table 5.5. Observed bugs that are predicted by the model as a percentage of all predicted observed bugs. The rank number is from table A.2 on page 188.

Observed bugs

The results so far indicate that the model, as measured by the number of star bugs, has little predictive power. The situation is much worse when the predicted observed bugs are considered (those bugs that the network exhibited and are also listed in the appendices). The model only predicted 6 different bugs. These are shown in table 5.5.

It should be noted that the model is incapable of capturing a number of bugs because of certain design decisions. For example, the bug $N \times 0 = N$ cannot be accounted for because the presence of particular digits was not part of the input vector. The network was trained on a very limited set of skills, which mostly involved scanning down columns. This lack of experience means that it is very unlikely that the network could produce bugs like adds-disregarding-columns. However, there are only a small number of exceptions, and the 6 predicted bugs is a woeful 5.88 per cent of the 102 bugs recorded in the appendices.

The distribution of observed and unobserved bugs should be interpreted with caution. The comparison to errors made by children is difficult to assess because there is no way to know how frequently each kind of problem occured. The number of times a bug occurs depends on how many chances it has of occuring. The set of problems used in this model was not designed to match the problems frequency experienced by children. Hence, the results presented above cannot be directly comapred to results reported for children.

5.4 Analysis

Emprical adequacy is not the only measure of a model. Rather than disregard the model because of the poor fit to the data, or attempt to "tweak" the model to improve the results, it is better to look into the system to understand its behaviour. After analysing the network (see next section) it is clear that the choice of operators is not quite right, and these could be developed. The important point to note here is that arithmetic is a procedural skill. Whichever way one conceptualizes multicolumn arithmetic, the result is a procedural skill, with all the entailments of subprocedures and conditional branching. It turns out that the mistakes the network makes are very "procedural" in nature. The poor fit to the observed data, I claim, is not a feature of connectionism, but a result of selecting an inappropriate representation of the problem.

5.4.1 *Finite state machines*

Arithmetic can be viewed as learning to be a finite state machine, and recurrent networks can learn to be finite state machines, so it seems apt to analyse the network as a FSM. There are a number of ways of extracting a FSM from a network (Giles, Miller, Chen, Chen, Sun & Lee 1992; Maskara & Noetzel 1992). The process starts by recording the hidden unit activations as a network solves a number of problems. The hidden unit vectors are then assigned to certain states in the FSM. Different methods use different ways of assinging hidden vectors to states. The Maskara & Noetzel approach is to assume that a particular hidden vector represents the same state as another if the corresponding elements from each vector differ by less than some specified tolerance. Giles et al. quantize every element of every hidden vector. The simplest case is to assume a binary quantization, in which each bit of the hidden vector is 1.0 if the activation is greater than 0.5, and zero otherwise. With three quantization states a hidden unit will be set to zero if its activation is less than 0.33, 1 if between 0.33 and 0.66, and 2 otherwise. The hidden vectors can be compared once they have all been quantized. Obviously the quantization size is the paramter equivalent to Maskara & Noetzel's tolerance value.

Figure 5.8 shows the FSM extracted from the first network, which had only been trained on 1+1. Hidden activations were recorded for the problems 1+1+1, 1+1+1+1, and 1+1+1+1+1. The states were grouped using Maskara & Noetzel's method with a tolerance of 0.04. The solution paths are identical up to state 9 (bottom centre of figure). For 1+1+1, the next thing the network has to do is move down once more, passing the rule mark,

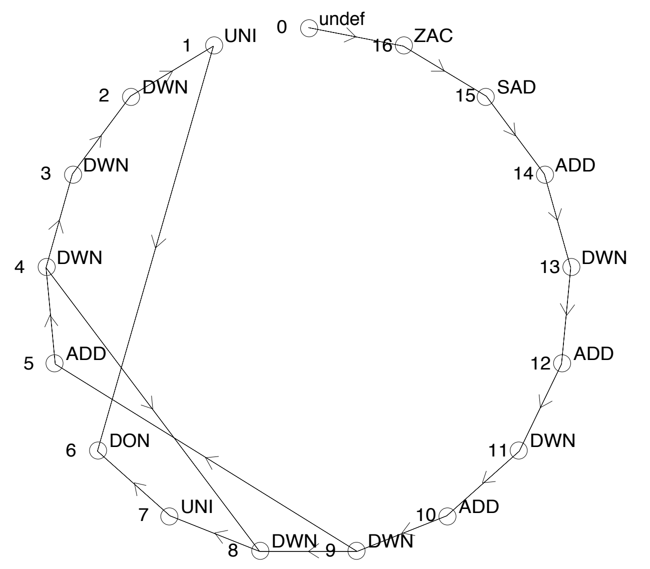

Figure 5.8. Finite state machine extracted from a network using the Maskara & Noetzel method (tolerance=0.04). The circles represent states, and the state numbers next to them are for reference purposes. Each state has a label associated with it (e.g., `DWN` for state 13). These are abreviations of the output operations listed in table 5.1

write the units digit, 3, and decalre itself done. For 1+1+1+1, the network must add the extra digit, before moving down and returning to the "write units and done" sequence. This particular network gives the answer of 4 for 1+1+1+1+1, and the reason for this can be seen in the figure. The behaviour changes at state 4 (centre left of figure), where instead of re-entering the add-down loop, the network executes three downs, skipping over the extra digit. It then writes the units and finishes.

The main difficulty in using FSM extraction as an anlysis tool is the setting of the similarity parameter—either the tolerance value or number of quantization levels. Setting the tolerance too low means that similar states are classified as different. A high tolerance can gives the impression that the network has generalized more than it actually has by classifying dissimilar states as a single state.

5.4.2 Graded state machines

Setting the similarity measure is a trial-and-error processes, and can give a false impression of the systems abilities. Also, the vary nature of FSM extraction means that hidden vectors either represent a particular state or they do not; there is no room for graded states (Servan-Schreiber et al. 1991). In reality, connectionist network are not finite state machines although they can act as if they were. Servan-Schreiber et al. (1991, p. 179) note:

> The network implementation [of a FSM] remains capable of performing similarity-based processing, making it somewhat noise-tolerant (the machine does not 'jam' if it encounters an undefined state transition and it can recover as the sequence of inputs continues), and it remains able to generalize to sequences that were not part of the training set.

Servan-Schreiber et al. consider these kinds of recurrent networks as "…an exemplar of a new class of automata that we call *graded state machines*" (p. 179). Rather than use cluster analysis (as Servan-Schreiber et al. do) or FSM extraction to analyse the behaviour of these systems, I find it more profitable to look at processing trajectories. This method begins with the recording of hidden unit activations, just as it does with FSM extraction. However, rather than computing states, the hidden vectors are plotted, preferably as points in 35 dimensional activation space. Each point is joined to the next in the sequence, striking out a solution trajectory.

To visualize these trajectories the dimensionality is reduced by performing principal components analysis (PCA). This is a method of computing the direction of greatest variation in the hidden unit activations. The original hidden unit vectors are projected back onto a selected number of the components of variation, usually two of the first three, giving a plot which (hopefully) has extracted the important aspects of the hidden unit representations. PCA has been widely used to analyse hidden layer representations (Dennis & Phillips 1991), but Elman (1989b, 1989a) was one of the first to use it to draw PCA *trajectory* graphs for sequential networks. Jordan (1986) also discussed attractor dynamics in sequential networks, but without looking into the hidden unit representations. Describing network behaviour in terms of PCA trajectories also invokes the language of dynamic systems (Cummins 1993).

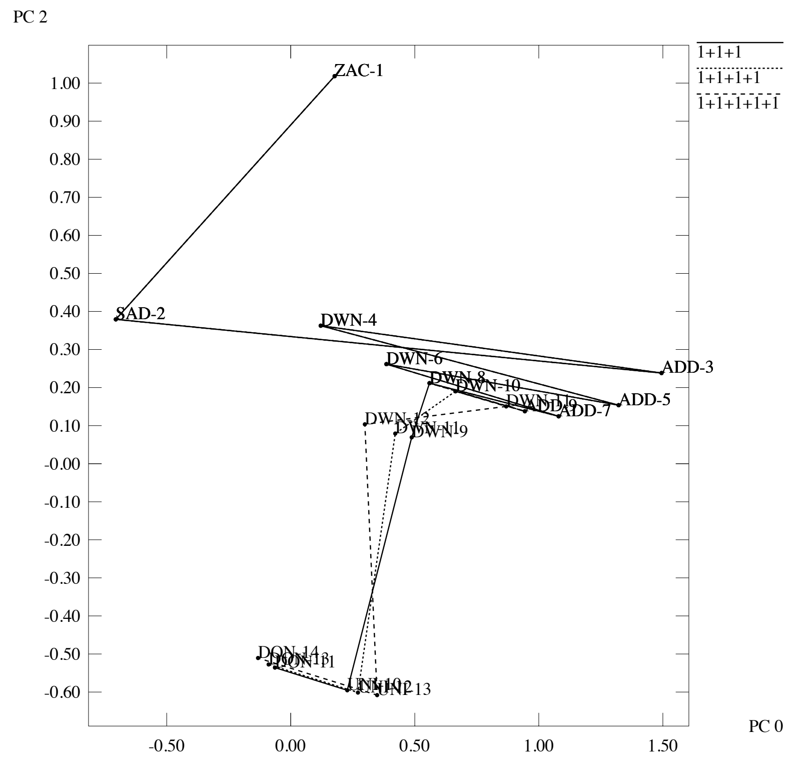

Figure 5.9. Trajectory of a network solving three addition problems. The trajectory directions are implied by the time step numbering.

Figure 5.9 shows the PCA trajectory for the network used in the FSM extraction of figure 5.8. Again, the hidden unit activations were recorded for 1+1+1, 1+1+1+1, and 1+1+1+1+1. The labels on the figure refer to the output action associated with a given point, and the number is the time step at which each of the actions occurred. For example, `ADD-3` refers to the action "add to accumulator" which occured on the third step in the sequence.

On the third problem the network fails to add the final 1. The first step is exactly the same for all three trajectories (zeroing the accumulator, `ZAC-1`). As with the FSM analysis, the PCA analysis shows that the problems follow the same trajectory up to step 7, where the trajectories are zigzagging between `DWN` and `ADD`. After the additions have been performed, all the trajectories emerge to perform the necessary two `DWN` actions and the final `DON`.

Although no explicit clustering of states has been performed, it is clear that regions of the PCA graph show something equivalent to states—e.g., the cluster of `DON` or `DWN` at

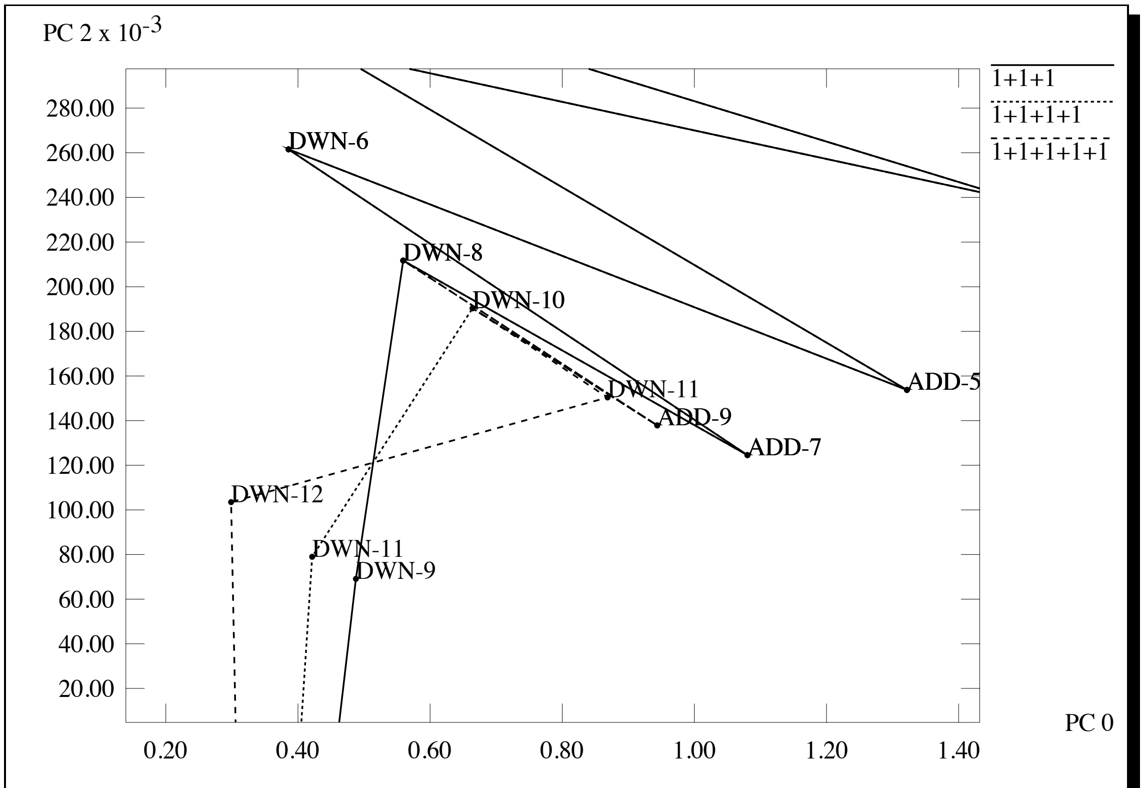

Figure 5.10. Zoom on the central region of figure 5.9.

the bottom of the figure. Zooming into the zigzag region gives a clearer picture of what is happening when the network incorrecty answer 1+1+1+1 (figure 5.10). The trajectories are identical upto `ADD-7`. The trajectory for 1+1+1 (solid line) moves to `DWN-8` and then heads off finish of the problem. For 1+1+1+1 and extra add-down sequence is required (`ADD-9` and `DWN-10`). The next step for 1+1+1+1+1 should be another addition, `ADD-11`, but as was shown earlier, the network skips over the digit with `DWN-11`.

Using PCA trajectories the explanation why the network makes this mistake is much clearer than that supplied by the FSM graph. Either the "add" attractor is not yet stong enough, or the transition out of the "down" attractor is not accurate enough. This kind of analysis and explanation is very useful in understanding why the networks make the mistakes they do, and what kinds of mistakes they could make. Other examples are given in this section.

5.4.3 Repairs

When the system encounters an unseen problem it does not halt. The actions it performs depend on the structure of the trajectory space, which is determined by the training sequence. As no explicit impasses occur within this system, there can be no repairs. However, the closest equivalent to a repair is the state transition that occurs at the point where the network encounters a new situation. This kinds of "repair" often takes the form of a divergence from the desired trajectory.

An example of the "repair as trajectory divergence" is given in figure 5.11. The graph contains the trajectories for three problems: 11+11, which the network solves correctly; 11×1, also solved correctly; and 11×11, which the network cannot solve. The graph was produced from a network which has been trained on two column by one column multiplications, but this is the first time it has encountered a two column by two column multiplication. However, the system has been trained on two column by two column addition.

All the trajectories follow a more-or-less anticlockwise spiral from the centre of the graph. It is worth tracing out the paths of the two multiplications (dashed lines) from `SMU-1` to see that, although they follow a similar path, they have diverged by `JTR-9` (bottom centre of graph). The divergence, which began after the first step, is due to the fact that the system is receiving different input patterns because the first column of the second row of 11×11 is not the leftmost column as it is for 11×1.

After the `JTR-9` step, both multiplication paths move on to correctly perform a multiplication, `MULT-10`. However, the next step for 11×1 is `JAS-11` (top of graph). For 11×11 the system should also jump to the answer space to write the product, but it has never been in a situation where there are two multiplicands and two multipliers. The system actually performs `DWN-11` and `ADD-12` which are very close together on the graph, just below `JAS-11`. The system then does the `JAS` operations and finished the problem as if it were 11×1.

This bug shows itself on 43×11. 43×11 produces the same sequence of actions as 11×11, but because different numbers are used it makes it clearer to see what is happening.

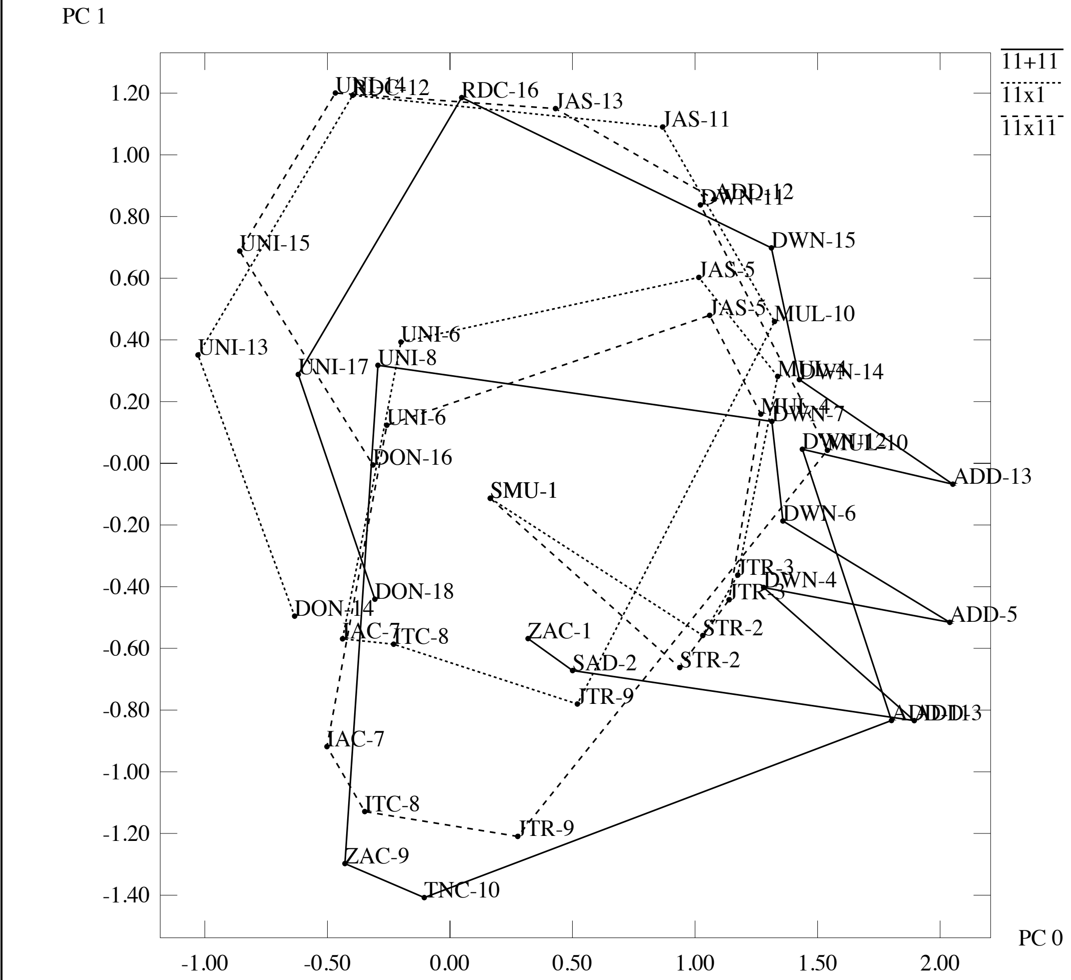

Figure 5.11. PC space trajectory for a network trained upto and including 1×11. This is problem 17 from table 5.3.

$$\begin{array}{r} 4\ 3 \\ \times\ \ 1\ 1 \\ \hline 5\ 3 \end{array}$$

After 1×3 is correctly answered, 1×4 is performed, then the "down" and "add" actions are executed giving an answer of 5. It seems that the system was attracted into part of the addition sequence, but then pulled back onto the multiplication path. It is not surprising that in a novel situation the system should resort to solving the problem using steps from a similar, trained solution (by following some of the steps from the trained two column by two column addition on an unseen two column by two column multiplication). What is unexpected is the fact that the system can "recover" and return to the multiplication path. This is possible because the network is not a memoryless FSM, but a graded state machine (GSM). The current state of the system can include information about the path taken. Servan-Schreiber et al. (1991) found this by cluster analysis: the major branches in the cluster were indeed the states of the FSM; however, further subdivisions were noted which individuated each path. In figure 5.11 this can be seen as the close, but not identical, positioning of states such as `JAS-11` and `JAS-13` at the top of the figure. Or again in figure 5.9, where there is a cluster of `DON` and `DWN` actions at the bottom of the figure, not a single point. In other words, GSMs differ from FSMs in that the next state is not just a function of the previous state and the current input, but can also be based on the path taken to reach the current state.

In a production system the above behaviour of mixing some addition in with the multiplication could be captured with a rule such as:

```
bugCC:  [no_more_top] ⇒ [startadd]
```

(See table 4.1 on page 91.) However it is not clear how or why such rules would be acquired given that `no_more_top` is specific to multiplication. With the trajectory analysis there is no such problems because all trajectories are represented within the same space. Observed bugs will just be confused trajectories. There is also an account of the development of bugs which does not have the snapshot nature of symbolic accounts (see section 5.4.5).

5.4.4 Accuracy of transitions

When training sequential networks, one usually thinks of having to learn the input/output mappings one step at a time—learning step n before learning $n + 1$, and then learning the other steps in sequence. In some case, however, the current training set may use “subroutines” that already exist in the trajectory space. This means that learning is a matter of adjusting existing transitions or creating new ones. The analogy to production systems might be the construction of new clauses in the conditions of rules.

For example, after training a network to solve 11×11, testing on 12×15 produces the bug quits-after-first-multiplier:

$$\begin{array}{r} 1\;\;2 \\ \times\;\;1\;\;5 \\ \hline 6_{1}\,0 \end{array}$$

The problem here is that at step 16 of the sequence the network signals that it has finished by activating the `DON` unit instead of starting the next answer row (`NAR`). This network has already been trained on a lot of problems, including 11×11. Hence, the subroutine for processing the second multiplier aready exists in the network, and it is just a matter of training the system to get the correct transition at step 16 of the problem.

This can be shown in the following way. Solving 12×15 requires 58 output actions. However it can be correctly solved by training on just the first 16 steps up to the point where the network is making the incorrect transition. This training adjusts the transitions between states to push the trajecetory on to the path for processing the second multiplier. After training, the network correctly performs steps 17–58 even though it was not trained on them explicity. The last 42 steps of the sequence come for free as a result of previous training on other problems.

Getting step 16 correct is not enough to solve the problem. At some time during the training of the 16 steps, the network produces the correct output at step 16 (i.e., `NAR`). This does not mean that the whole sequence of 58 steps is correct. Rather, the network will produce a few more steps of the sequence, but training on the 16 steps must continue for some time before the sequence is fully learned. Although the output is correct, the transition needs to be fine-tuned to ensure enough information is preserved to allow the system to follow the correct trajectory.

Another situation in which accuracy is important is demonstrated by the bug shown on page 134, where 118+590 is solved correctly alone, but not when part of a multiplication. Here the problem is that one transition is not accurate enough in the context of the many preceeding steps. A single step, marking the carry, is skipped over as a result. Sleeman (1982) found a number of examples of these context-dependent errors in algebra protocols: "…when the problem is 'hard', the student makes errors with rules with which he previously suceeded" (p. 198). It seems that these errors are difficult to model (for Sleeman's system).

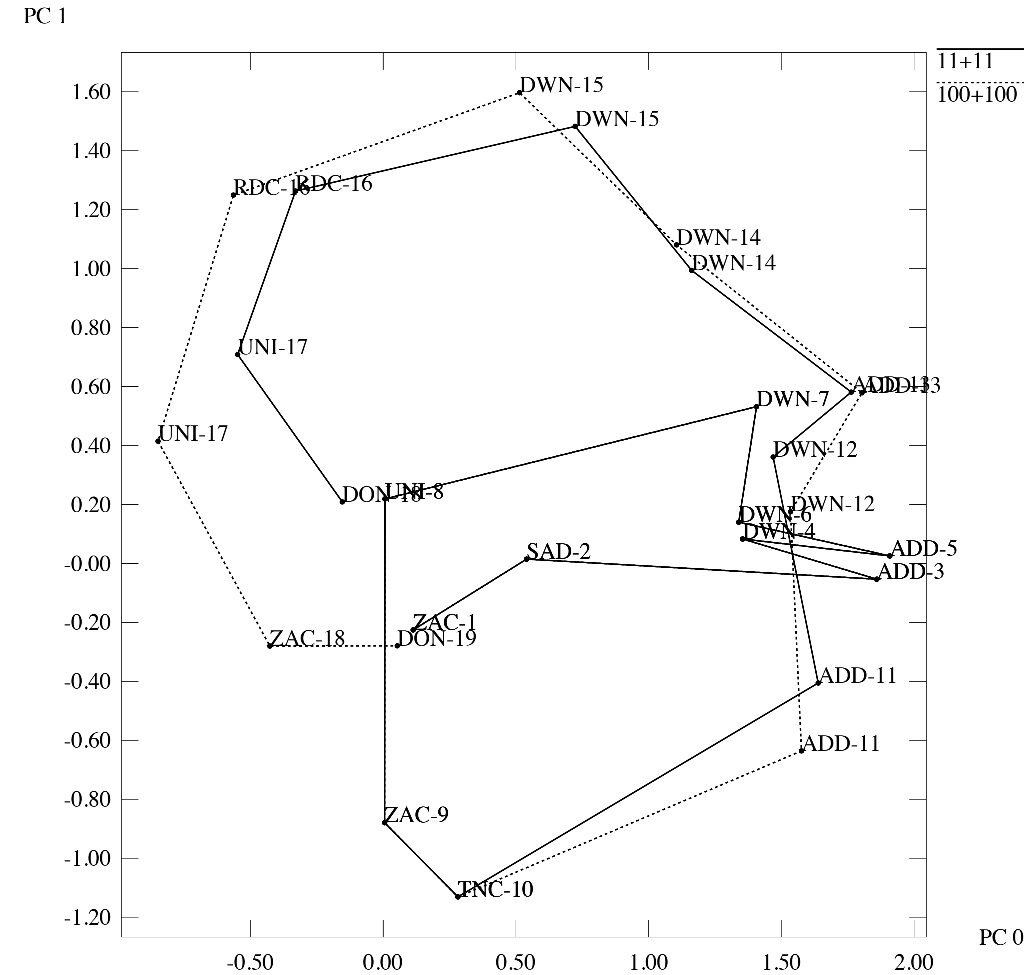

Figure 5.12. Trajectories for the problems 11+11 (correct) and 100+100 (incorrect).

5.4.5 Development of trajectories

One of the more appealing aspects of connectionism is that the weights of the system change slowly, allowing the model to escape the snapshot nature of symbolic models in this domain. Learning, according to the analysis shown in this chapter, is about getting the right transitions between states. States are not single points in a space, but are clustered in an area. So in addition to learning and tuning transitions, the system needs to expand or contract attractors for certain states to ensure that trajectories fall in just the right places as defined by the training set.

Figure 5.12 and figure 5.13 show a network before and after training on the problem 100+100 (three column addition without carrying). Both systems correctly solve 11+11,

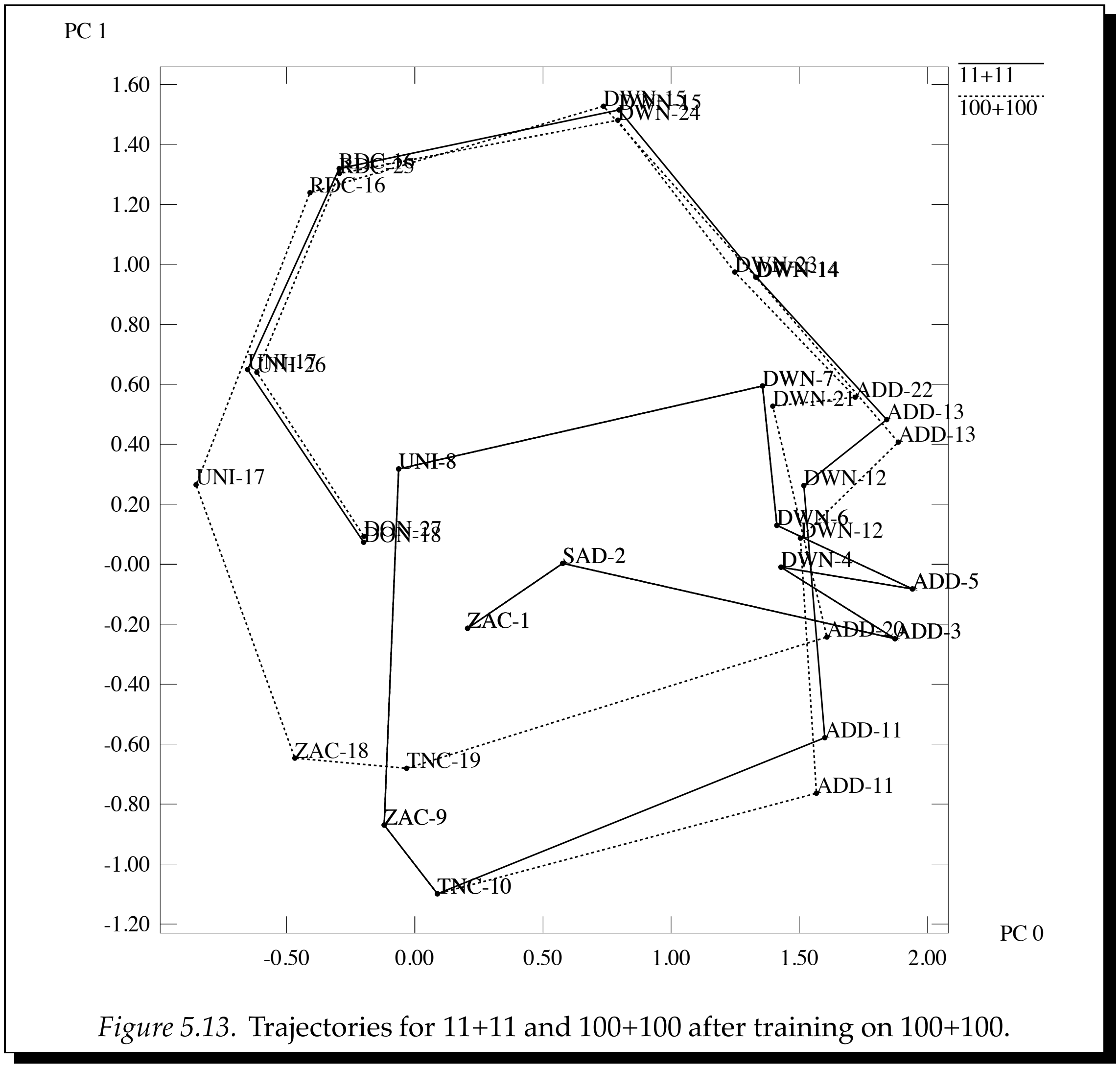

Figure 5.13. Trajectories for 11+11 and 100+100 after training on 100+100.

but only the second solves 100+100. In the case of figure 5.12, the behaviour of the system is to simply ignore the third column:

$$\begin{array}{r} 1\ 0\ 0 \\ +\ 1\ 0\ 0 \\ \hline 0\ 0 \end{array}$$

Figure 5.12 shows that both problems strike out the same trajectory up to `TNC-10` (centre bottom of the figure). At this point the first column has been solved. Both problems then follow similar paths to process the second column. For the incorrect trajectory, after writing the answer to the second column, the system correctly resets the accumulator (`ZAC-18`), but then finishes (`DON-19`). The desired behaviour is for the system to follow the "process a column" loop one more time. This can be achieved if the `ZAC-18` step was moved closer to `ZAC-9`. After learning this is exactly what is seen (figure 5.13).

5.4.6 Summary

An analysis has been presented of the system's behaviour in terms of solution trajectories in principal components space. This kind of analysis is not without its problems. Most notably is the problem of projecting a 35 dimensional hidden unit activation space to the two or three dimensions that can be visualized. As the network requires 35 hidden units to solve the problem there is no reason to believe that two dimensions are going to be adequate to understand the system. However, the usefulness of the method is an empirical matter, and it seems that there is an interesting story that can be told about the system by looking at trajectories in PC space.

The view presented in this chapter is that:

- Learning is the gradual adjustment of state attractors and state transitions. Once enough states have been distinguished between, learning is a matter of getting the correct state transitions.
- Bugs are incorrect transitions due to overlapping states.
- Repairs, in the sense of doing something sensible in novel situations, are blended trajectories that result from generalization and similarity based processing.
- Impasses do not occur.

Although the results of simulations do not fit the empirical data very well, it can be said that the network can capture certain kinds of behaviours that are appropriate for modelling arithmetic. The analysis has shown that the trajectory space of the system can contain regions that correspond to subskills—e.g., for processing a column. These subroutines can be entered into and returned from because the network is not a FSM, but a graded state machine, allowing states of a trajectory to have a memory of their path. Such subroutines may be useful during learning, whereby the network does not have to relearn a certain behaviour but can modify transitions into and out of a sequence of actions.

It seems that an inappropriate set of operations were devised for this model. However, *whichever way arithmetic is sliced, the result is a procedural skill.* That is, a skill made up of subskills which are executed at the right moment. Production systems model "the right moment" in the condition parts of the rules, and execute subroutines on the action part of the rules. For the network, subskills are the well-worn trajectories, entered into by (sometimes inappropriate) state transitions. Given the previous work on production system models, this seems sufficient to capture the kinds of errors seen in arithmetic.

The explanation of errors is similar in spirit to the Norman's "capture errors"…

> …in which a frequently done activity suddenly takes charge instead of (captures) the one intended. You are playing a piece of music (without too much attention) and it is similar to another (which you know better); suddenly you are playing the more familiar piece…Or you get into your car on Sunday to go to the store and find yourself at the office.
>
> The capture error appears whenever two different action sequences have their initial stages in common, with one sequence being unfamiliar and the other being well practiced.
>
> (Norman 1988, p. 107)

Norman classed these errors as slips, whereas the errors described here are procedural.

5.5 Bug migration

The system as presented so far always produces the same bug when run on the same problem. Yet as discussed in chapter 4 children switch between bug sets over long and short periods of time. The phenomenon, called bug migration, can be captured by the network in two ways: by adding noise during processing; or as a result of the slow changes to the weights.

5.5.1 Bug migration as noise

Processing trajectories can be perturbed by adding noise to the activations of processing units. There are many ways to do this. The results presented in this section are based on adding noise to each connection, with the noise proportional to the magnitude of the weight. Specifically, the net input to a non-input unit was changed to:

$$\text{net}_i = \sum_j (a_j w_{ij} + h(\frac{w_{ij}}{30}))$$

where $h(n)$ returns a random number between $\pm n$. There was no particular reason for using this method, and I suspect that other ways of adding noise would produce similar results. The particular value of a thirtieth of the weight was selected by trial and error so that the networks would produce a variety of behaviours, but still be able to produce the behaviour they would if noise was not present.

Testing a network with this modified net input function produces a number of different behaviours for the same problem. Hence, testing was as follows. The 27 networks used in the experiments of section 5.3 were presented with the next unseen problem in the curriculum. This same problem was presented 20 times in order to record the distribution of behaviours.

Fifteen networks exhibited no variety in their behaviour, and just produced the behaviour that they would without noise. Five networks produced 2 or 3 behaviours. The remaining 7 networks produced: 7, 8, 9, 12, 13, 15, and 17 behaviours.

The network that produced 7 behaviours was trained on 11×1 and tested on 111×1. The behaviour without noise for this network was to process the first two columns correctly, and then write the product of the third column (1) in the problem, producing:

$$\begin{array}{cccc} & 1 & 1 & 1 \\ \times & 1 & & 1 \\ \hline & & 1 & 1 \end{array}$$

This behaviour was also produced on 11 out of 20 of the runs with noise. The network entered into a infinite loop on 2 runs, and produced the following 2 behaviours on another 2 occasions:

$$\begin{array}{cccc} & 1 & 1 & 1 \\ \times & & & 1 \\ \hline & 1 & 1 & 1 \end{array} \qquad\qquad \begin{array}{cccc} & 1 & 1 & 1 \\ \times & & & 1 \\ \hline & & 1 & 1 \end{array}$$

Notice that the first of these behaviours is the correct answer. Finally, 3 behaviours

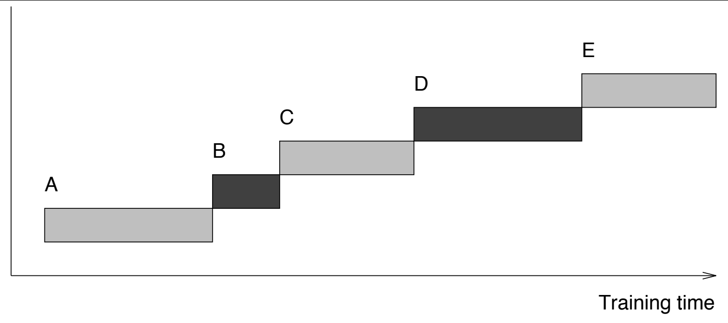

Figure 5.14. Representation of the amount of time in epochs that five behaviours persisted. See text for explanation.

occurred only once in the 20 runs:

$$\begin{array}{rrrr} & 1 & 1 & 1 \\ \times & & 1 & 1 \\ \hline & 1 & 1 & 1 \end{array} \qquad \begin{array}{rrrr} & 1 & 1 & 1 \\ \times & & & 1 \\ \hline & & 2 & 1 \end{array} \qquad \begin{array}{rrrr} & 1 & 1 & 1 \\ \times & & & 1 \\ \hline & 2 & 1 & 1 \\ & 2 & & \end{array}$$

The majority of these behaviours constitute star bugs, but this is not surprising given that the system as a whole produces a large number of star bugs. Allowing noise into the system, causing different behaviours to occur with varying frequencies, is one way bug migration could be accounted for.

5.5.2 *Bug migration as learning*

The three repairs that VanLehn proposes are applied to impasses randomly. To account for bug stability, he allows "patches" to be made to the local problem solver. This connects particular impasses with particular repairs, allowing a behaviour to persist for some time.

The account of "bug migration as noise" does not suggest any consistency in observed bugs. However there is a way for a connectionist model to account for bug migration, and also allow the bug set to change over varying periods of time.

During learning, the behaviour of the network will change. Bug migration can be accounted for by assuming that learning is a continuous process—not "a thing" that happens at a particular moment.

For example, figure 5.14 shows the amount of time a particular behaviour is observed during learning. In this case the network is learning the problem 11×1, and being tested on 234×2. The behaviours, all star bugs, are:

A: $$\begin{array}{cccc} & 2 & 3 & 4 \\ \times & 4 & & 2 \\ \hline & & 6 & 8 \end{array}$$

B: $$\begin{array}{cccc} & 2 & 3 & 4 \\ \times & 4 & 6 & 2 \\ \hline & & 6 & 8 \end{array}$$

C: $$\begin{array}{cccc} & 2 & 3 & 4 \\ \times & & 6 & 2 \\ \hline & 1 & 6 & 8 \end{array}$$

D: $$\begin{array}{cccc} & 2 & 3 & 4 \\ \times & 4 & & 2 \\ \hline & & 4 & 8 \end{array}$$

E: $$\begin{array}{cccc} & 8 & 3 & 4 \\ \times & & & 2 \\ \hline & & 9 & 8 \end{array}$$

As training continued on 11×1, the network was tested on 234×2 every 1500 epochs. In the figure, behaviour A persists for 7500 epochs, and B for 3000 epochs. Hence, some behaviours may exist for a relatively long time, while others are short-lived. So although the weights are always changing (continuous quantitative change), particular behaviours can survive for varying periods of time (sporadic qualitative change).

5.6 Impasses

Impasses are:

1. Detectable by the processor solving the problem.

2. Characterized by pauses or negative comments, such as "I don't know what to do".

3. Possibly the point at which learning takes place (VanLehn 1988, 1991a).

By this definition the connectionist model does not have impasses. However, the networks show some interesting behaviours at the moments when new situations are encountered.

In chapter 3, the experiments on memory for multiplication facts required the measurement of a reaction time. The activation rule of the network was changed to allow activation to build up slowly. The RT measure was the number of cycles needed for an output unit to reach a threshold value.

It turns out that RT, when measured in this way, is correlated to the error on the output layer. This should not be surprising because as the network learns the error is reduced, and RT decreases as was shown in chapter 3. This means that a rough measure of RT

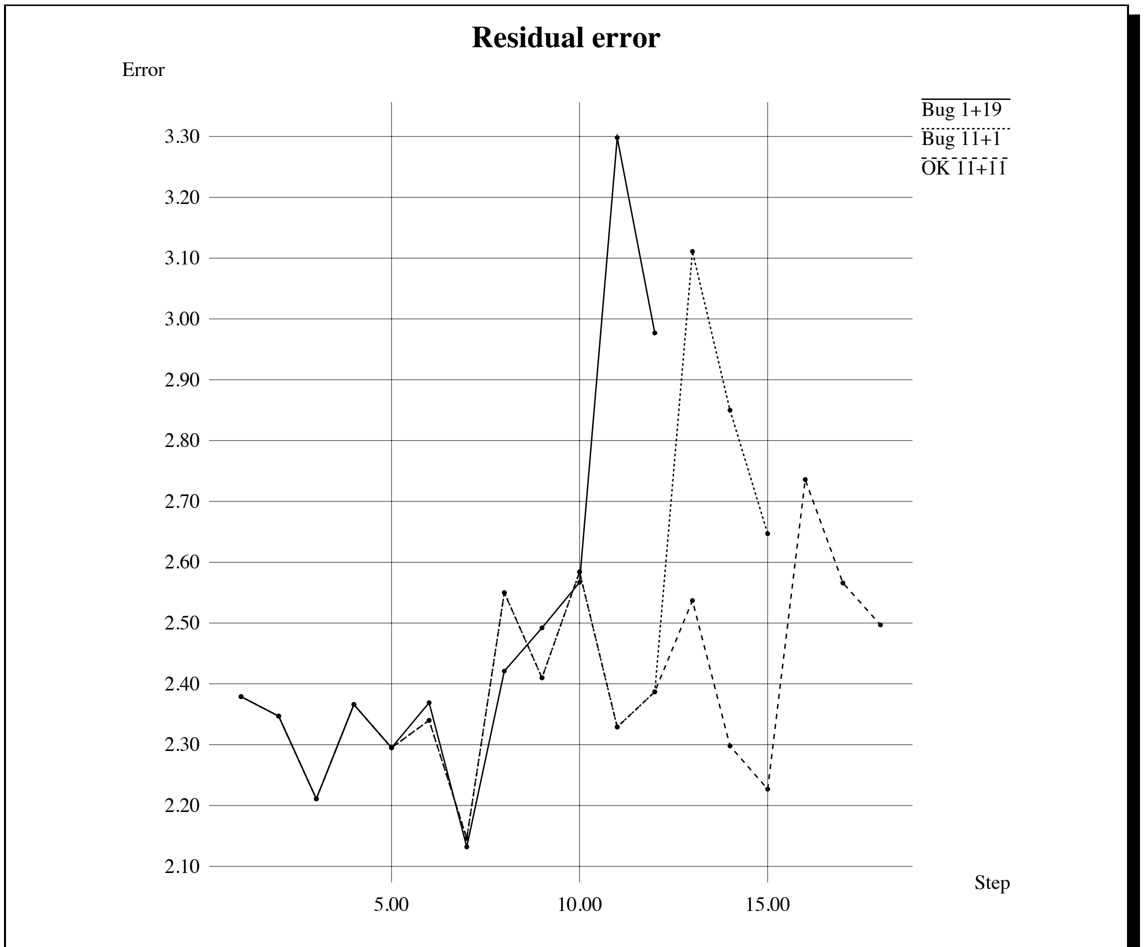

Figure 5.15. Residual error for a network solving three addition problems.

can be computed for the recurrent network introduced in this chapter, without having to install the machinary needed to record RT, such as a "don't know" unit.

Given the way the system works, there are some situations where there is no target output. For example, when the system is running on a novel problem, the behaviour may be buggy, resulting in a sequence of actions that may be longer than the target sequence. Hence the usual error measure of target activation minus actual activation is not used, and instead the "residual error" of the system is reported. Residual error assumes that one output unit—the one with the largest activation—should be on and the others should be off. Error is measured as the deviation from this one-of-N desired vector.

This error can be plotted as the network solves problems. For example, figure 5.15 plots the error for each step a network takes as it solves: 1+19, which is incorretly solved; 11+1, also wrong; and 11+11, which is correct. The network in question has been trained up to and including 11+11, but not on 1+19 or 11+1. In the case of 1+19, the network quits when it focusses on the empty cell above the 1. For 11+1, the network processes the first column correctly, and then writes the sum of the second column in the empty cell above the rule line:

$$\begin{array}{rrr} & & 1 \\ + & 1 & 9 \\ \hline & & 0 \end{array} \qquad\qquad \begin{array}{rrr} & 1 & 1 \\ + & 1 & 1 \\ \hline & & 2 \end{array}$$

In the figure, the peaks in the error plots for 1+19 and 11+1 correspond to “classic” impassess: the moment when the network has to deal with an empty cell. If the error measure is correlated to RT, then these are also the moments when the network takes the longest time to carry out an action, which might be called a repair. But of course these “impasses” are not explicitly recognized or utilized by the network, and as such are not “real impasses” at all.

Many of the bugs produced by the network occur towards the end of a sequence (as they do in figure 5.15). For longer sequences, priods of buggy behaviour can give way to relatively normal processing. Figure 5.16 shows such a sequence for a network trained on 12×19 and tested on 12×50. Here the “impasse” is when a carry arises from the second multiplier (5×2). The figure shows the residual error increase at this point, and remains high for some time. Towards the end of the sequence, during the addition phase, the error drops back down again.

It should be noted that these error peaks are generally the places where backpropagation does much of its work of reducing error. In this sense, learning is driven by pseudo-impasses, although clearly it is nothing like the impasse-driven learning proposed by VanLehn.

The point to note here is that processing in the network is not homogeneous, and different stages of a solution can cause the network some difficulty as measured by residual error. If the error was translated into reaction time, the network would have periods of slow processing. The RT mechanisms for these events require further elaboration and detailed simulations. It might be suppoosed that the RT mechanism is not dissimilar to that presented for the memory of arithmetic facts. This also suggests that impasses may

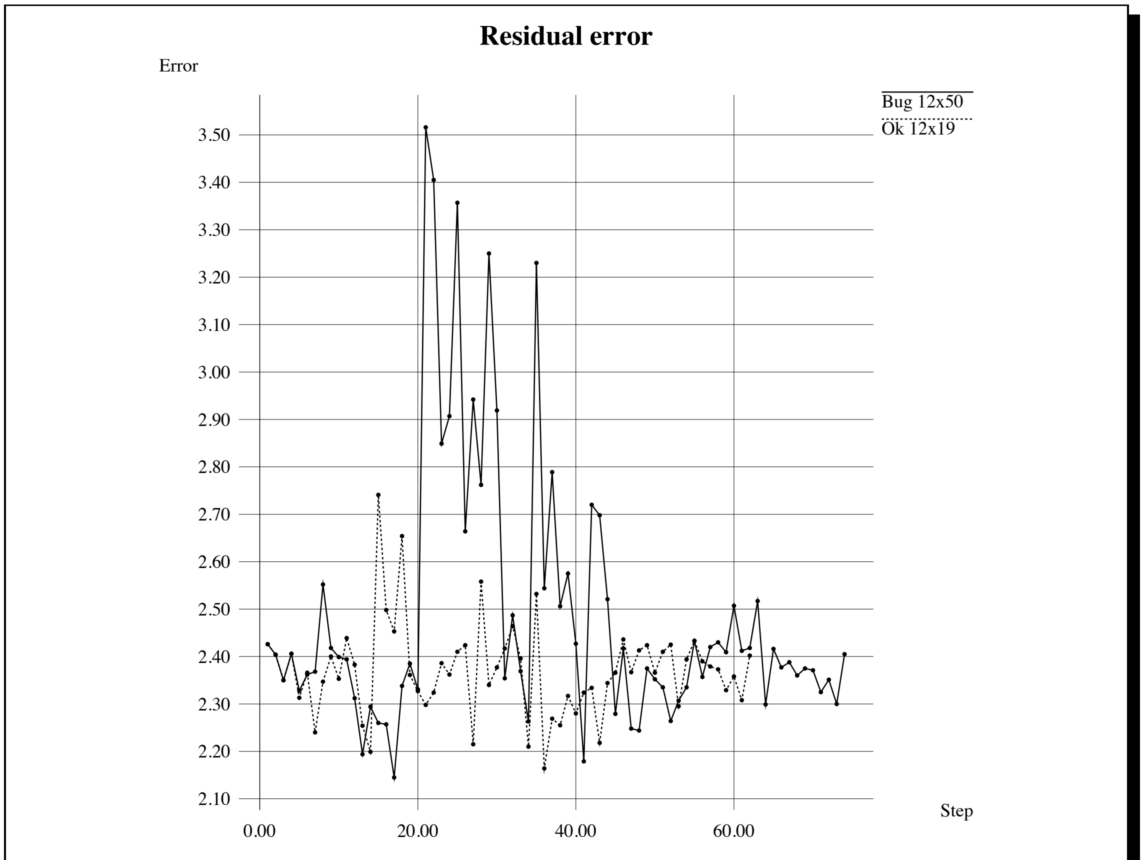

Figure 5.16. Residual error for problems 12×50 and 12×19.

not be all-or-none events, but a graded phenomena. It also seems likely that the moments of high error are most susceptible to peturbation by noise.

5.6.1 The importance of impasses

Systems like Soar (Laird et al. 1986) rely on impasses. However these impasses are different from the ones discussed by VanLehn. Soar's impasses are "little impasses" in that they occur very frequently (VanLehn 1991a). The "big impasses" of Sierra occur at a much larger scale (for a comparison of the two kinds of impasses see VanLehn 1990, pp. 59–61). This difference is because Soar is a more detailed model of cognition, including a model of memory. Other models, such as ACT* (J. R. Anderson 1983), do not have impasses at all. Hence the importance of impasses is an empirical matter.

It is worth looking at the appeal of impasses. VanLehn (1988) is developing a impasse-driven model of learning. The idea is that when an impasse is reached, some kind of learning occurs (e.g., by asking the teacher what to do next; see VanLehn 1990, p. 145). Impasse driven learning would also remove the need for some of the assumptions placed on VanLehn's learner.

Perhaps the most persuasive argument for the importance of impasses in learning comes from VanLehn's (1991a) study of "rule acquisition events". A protocol of a subject solving the Tower of Hanoi problem was analysed for the moments when new rules were acquired. That is, VanLehn constructed a set of rules to fit the steps in the protocol. Of the 11 rule acquisition events in the protocol, 8 occurred at an impasse. VanLehn reports that the remaining 3 had nothing to do with impasses at all. Although impasse-driven learning does not account for all the acquisition events in this case, it clearly could be a powerful mechanism.

Yet it is quite possible that impasses are just a symptom, not the cause, of change. This is the suggestion implied by the discussion of residual error: the role of such events is a secondary phenomena, one that is just the side-effect of the architecture. From another point of view, it might be said that although there is a correlation between impasses and learning events, the mechanisms have not been fully elaborated.

The safest conclusion at this point that there may be many kinds of impasses. VanLehn proposes that some are crucial to learning, where as I suggest that some may be inconsequential. In any event the fact that impasse behaviour is observed requires an explanation. Their importance is an empirical matter.

5.7 Summary

This chapter has shown that a connectionist system can "generate the kind of extended, sequential problem-solving behaviour that characterizes students solving subtraction problems". This was the original aim. There was clearly little hope that the model would match the empirical findings as well as Sierra does. Sierra, after all, grew out of more than ten years of research on the impasse-repair process.

Production system models are sucessful in this domain because arithmetic is a procedural skill. Despite initial assumptions about what connectionist networks can or cannot represent well, it seems that there is a great deal of structure in the hidden layer activations. It also seems, from the analysis presented above, that the representations are organized into subskills that can be utilized by the model. This property suggests that the model is capable of interesting sequential behaviour, and it also changes the way arithmetic problem solving is conceptualized.

It turns out, for example, that it is possible to model buggy behaviour without an *explicit* impasse and repair process. The repairs carried out by the network, if they can be called repairs, are just a product of the dynamics of the system. When a new situation is met the solution path depends not on handful of general purpose heuristics, but on the statistical distribution of paths that have previously been followed.

Noise can be introduced to the system to vary behaviour. A more interesting possibility comes from the idea that the processor (child or network) should *not* be considered as a static entity. Although this conceptualization is encouraged by symbolic (snapshot) models, it may be more profitable to think of the system as being constantly in flux. Connectionist models promote this view.

Once the idea of a continuously changing, similarity-based system is taken seriously, the purpose of an explict repair mechanism has to be questioned. This thought was the basis of the model described in this chapter.

Having demonstrated that the networks show increases in error at moments that might be classed as kinds of impasses, the obvious question to ask is: do impasses have an important role to play, or are they just a side-effect of the processor? Tests should be devised to determine which view of impasses is more appropriate.

This work can be continued in a number of ways. With the current arcitecture the training environment can be explored. For example, errors resulting from “missing knowledge” might be observed if a step in the training sequence was missed out—perhaps because the child was ill and missed a lesson. Combining the multicolumn network with the memory network would provide another source of errors, whereby recall slips lead the multicolumn system into capture errors. The model predicts an increase in capture errors when a new skill is introduced, such as learning multicolumn multiplication after addition. Whether this is true of Sierra or children is something that should be investigated.

Longer term goals might include looking at the contraints that can be placed on the model from brain-damage studies (McCloskey, Aliminosa & Sokol 1991). The model has placed much emphasis on the importance of understanding the perceptual basis of arithmetic. More empirical work is needed in this area. On a related point there is much work to be done on the set of primitive operations used by the network. Having shown that a network can represent procedural information in an interest way, it remains to be shown that system could fit the data. As an example of a possible change would be to break down the `mark_carry` operation into smaller operations. Without this it is not possible to model bugs like does-not-rename-sum. There are many possible representation schemes, and a huge number of potential operations; exploring the possibilities will require further emprical constraints on the model.

The central issue is of impasses. Perhaps the significants of impasses will become clear when impasse-driven learning models are constructed. However, this chapter has attempted to show how it might be possible to account for arithmetic bugs without the emphasis on impasses.

CHAPTER 6

Summary

There were two main aims in writing this thesis:

1. To build an explicitly specified model of adult memory for multiplication facts.
2. To demonstrate an alternative view of children's multicolumn arithmetic, and show that such an approach is useful.

6.1 Memory for arithmetic facts

Chapter 2 contains a review of the literature and previous models of adult memory for multiplication facts. It was noted that RTs tend to be lower for smaller problems than larger problems, although there are exceptions to this rule. Most errors are operand errors—answers that are correct for a problem that shares an operand with the presented problem. With the exception of McCloskey & Lindermann's MATHNET model, previous models lack details about learning, response mechanisms or the spread of activation.

The cascade model presented in chapter 3 was trained on the multiplication facts and captures the main aspects of the phenomena. Recall from the network is based on a build up of activation in the hidden and output units. The RT is measured as the number of processing cycles required before a product unit exceeds some randomly selected threshold. When the threshold is low, incorrect products can be selected as the answer. These errors are mostly operand errors.

Earlier experiments used different assumptions about representation of operands and the frequency with which problems occurred. These experiments, together with an analysis of the networks, suggest that the following factors contribute towards the problem-size effect and error distribution: variations in input representation, especially the relative "sharpness" of the encoding; how frequently each problem occurs in the training set; and the nature of the arithmetic facts themselves. It also was noted that coarse encoding is equivalent to training on false associations. Some models assume that false associations are learned, and some do not. This thesis indicates that the question of interest is not whether or not false associations are formed, but by which method they are formed.

Preliminary simulations were presented of network damage and recall of zero and ones problems. Finally, possible accounts of verification and priming were discussed.

The various network models show some degree of consensus regarding the phenomena associated with recall of arithmetic facts. For example, the problem is considered as the spread of activation between operand units and answer units. However, there are many interesting differences between the models, including: input and output representation, intermediate representations, activation rules, training assumptions. The importance of these differences remains to be explored. For example, one of the assumptions in the cascade model is that the presence of a tie problem is made explicit in the input to the network. For adults tie problems are solved quickly, but for the network, this is only achieved by the use of a tie flag. At the moment it appears that tie problems are difficult for network models to account for without a measure equivalent to the ties flag.

6.2 Multicolumn arithmetic

Chapter 4 described children's errors in multicolumn multiplication. Previous accounts of buggy behaviour were considered—especially VanLehn's Sierra model. Sierra is an extension to repair theory and includes an inductive learning mechanism. VanLehn's model predicts that when children reach impasses, general purpose repairs are made to the local problem solver. The errors that are observed depend on what kind of impasse occurred and which repair was carried out.

A number of observations were made of why connectionism can contribute to this domain. It was noted that the notion of a impasse does not directly apply to connectionist networks: given an input, the network will produce an output. Networks may be able to automatically repair undefined situations because of such properties as similarity-based processing and automatic generalization.

Using some of the assumptions of VanLehn, and taking ideas from Suppes et al.'s model of eye-movement, a connectionist model of multicolumn multiplication was built (chapter 5). To study bugs, rather than slips, the network was trained to activate procedures to carry out the details of multiplication, such as adding, multiplying, and keeping track of registers. The recurrent network was trained on problems of ever-increasing difficulty, from 1+1 to 12×99. During training, the network was tested on unseen problems from the curriculum, and errors occurred at this point.

The errors made by the system do not match the empirical observations very well, although there are difficulties in comparing the errors to children's errors. The set of output operations, although sufficient for solving multiplication problems, requires further work to capture children's errors in detail. An analysis of the errors made by the network shows some interesting results. The system is behaving as a graded state machine: it has many of the properties of finite state machines, but does not "get stuck" when encountering novel inputs. Errors were characterized as perturbations to the desired trajectory, rather than perturbations to a rule set. The errors are a result of unlearned state transitions, and the details of a particular error depends on its similarity to previous experienced problems.

The state of the network was visualized by plotting the principal components of the hidden unit activations. Although it is not obvious that this reduction in dimensions (from 35 hidden units to 2 axis) will provide any interesting information about the system, in practice it does. The mistakes made by the system are capture errors: the system is temporarily attracted into a region of state space which represents an arithmetic subroutine. This is clearly visible with the PCA trajectory diagrams.

The representations learned by the system have a great deal of structure. The model suggests that this may be exploited to account for errors without reference to explicit impasses or repairs. It was noted that the output layer of the network exhibits an increase in residual error at moments that correspond to impasses. If the processing details of the network were changed, this increase in residual error could be observed an increase in RT. This raises the question of whether impasses are important moments for the learner, or simply a by-product of the processing mechanisms. The model requires more work before this suggestion can be more thoroughly explored and tested.

This is, of course, just one of many possible connectionist views of impasses. From the point of view of Soar, for example, Rosenbloom (1989) suggests that connectionist impasses may occur when a number of output units are above threshold—meaning that there is no uniquely specified course of action to take.

6.3 Future work

Specific future work, in the short- and mid-term, was outlined at the end of chapters 3 and 5. More general comments are made here.

Our understanding of the representation of number and of the training environment is poor. In both models the training environment—frequency or order of problems—is important. Empirical evidence needs to be accumulated to understand what problems children actually encounter.

Experiments in part I showed that changes in the "sharpness" of operand representation changed the results of the simulation. In part II, emphasis was placed on arithmetic perceptual skills, and in particular on eye-movements. Without an understanding of these details it will be difficult to build an appropriate operation set for the multicolumn model. It seems that more study is needed of preschool number abilities and foundational skills, such as number comparison and counting. The representation of number assumed and developed by the recall network should be applied to these other number skills. In this way it may be possible to determine the validity of the various representations, and evaluate the plausibility of a product level of representation and a tens and units level.

Bibliography

Ainsworth, S. (1991). A model of children's multiplication skills. Unpublished project, Portsmouth Polytechnic, UK.

Allen, R. B. (1988). Connectionist state machines. Technical report ARA 88-300, Bellcore, Morristown, NJ.

Allen, R. B. (1990). Connectionist language users. *Connection Science*, *2*(4), 279–311.

Anderson, J. A., Rossen, M. L., Viscuso, S. R. & Sereno, M. E. (1990). Experiments with representation in neural networks: object motion, speech, and arithmetic (Including introduction by Anderson). In Anderson, J. A., Pellionisz, A. & Rosenfeld, E., eds, *Neurocomputing 2: Directions For Research*, chapter 14, pp. 704–716. MIT Press, Cambridge, MA. First published in H. Haken and M. Stadler (Eds.) *Synergetics of Cognition*, Springer, Berlin.

Anderson, J. A., Spoehr, K. T. & Bennett, D. J. (1991). A study of numerical perversity: teaching arithmetic to a neural network. Technical report 91–3, Department of Cognitive and Linguistic Sciences, Brown University. To appear in Levine, D. S. and Aparicio, M. (eds) *Neural Networks for Knowledge Representation and Inference*, Lawrence Erlbaum Associates, Hillsdate, NJ.

Anderson, J. R. (1983). *The Architecture of Cognition*. Harvard University Press, Cambridge, MA.

Ashcraft, M. H. (1982). The development of mental arithmetic: a chronometric approach. *Developmental Review*, *2*, 213–236.

Ashcroft, M. H. (1987). Children's knowledge of simple arithmetic: a developmental model and simulation. In Bisanz, J., Brainerd, C. J. & Kail, R., eds, *Formal Methods in Developmental Psychology*, chapter 9, pp. 302–338. Springer-Verlag, New York, NY.

Ashcroft, M. H. & Stazyk, E. H. (1981). Mental addition: a test of three verification models. *Memory and Cognition*, *9*(2), 185–196.

Attisha, M. G. (1983). A microcomputer based tutoring system for self-improving and teaching techniques in arithmetic skills. Master's thesis, Faculty of Science, University of Exeter.

Attisha, M. G. & Yazdani, M. (1984). An expert system for diagnosing children's multiplication errors. *Instructional Science*, *13*, 79–92.

Bates, E. A. & Elman, J. L. (1992). Connectionism and the study of change. Technical report CRL 9202, Center for Research in Language, University of California, San Diego, La Jolla, CA. To appear in M. Johnson (Ed.) *Brain Development and Cognition: A Reader.* Oxford: Blackwell Publishers.

Boden, M. A. (1988). *Computer Models of Mind*. Cambridge University Press, Cambridge, UK.

Brown, J. S. & Burton, R. R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science*, *2*, 155–192.

Brown, J. S. & VanLehn, K. (1980). Repair theory: a generative theory of bugs in procedural knowledge. *Cognitive Science*, *4*, 379–426.

Burton, R. R. (1982). Diagnosing bugs in a simple procedural skill. In Sleeman, D. H. & Brown, J. S., eds, *Intelligent Tutoring Systems*, pp. 157–183. Academic Press, London.

Buswell, G. T. (1926). *Diagnostic Studies in Arithemetic*. University of Chicago Press, Chicago, IL.

Campbell, J. I. D. (1987). The role of associative interference in learning and retrieving arithmetic facts. In Sloboda, J. A. & Rogers, D., eds, *Cognitive Processes in Mathematics*, pp. 107–122. Clarendon Press, Oxford.

Campbell, J. I. D. & Graham, D. J. (1985). Mental multiplication skill: structure, process, and acquisition. *Canadian Journal of Psychology*, *39*(2), 338–366.

Chalmers, D. J. (1990). Syntactic transformations of distributed representations. Technical report, Indiana University. To appear in *Connection Science*, volume 2.

Churchland, P. S. & Sejnowski, T. J. (1989). Neural representation and neural computation. In Nadel, L., Copper, L. A., Culicover, P. & Harnish, R. M., eds, *Neural Connections, Mental Computation*, chapter 1, pp. 15–48. MIT Press, Cambridge, MA.

Clark, A. (1985). Artificial intelligence and the biological factor. Technical report CSRP 49, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Clark, A. (1986). A biological metaphor. *Mind & Language*, *1*(1), 45–63.

Clark, A. (1987a). Connectionism and cognitive science. In Hallam, J. & Mellish, C., eds, *Advances in Artificial Intelligence*, pp. 3–15. John Wiley and Sons, Chichester.

Clark, A. (1987b). The kludge in the machine. *Mind & Language*, *2*(4), 277–300.

Clark, A. (1989). *Microcognition: Philosophy, Cognitive Science, and Parallel Distributed Processing*. MIT Press, Cambridge, MA.

Clark, A. (1993). *Associative Engines: Connectionism, Concepts and Representational Change*. MIT Press, Cambridge, MA. To appear in 1993.

Cleeremans, A., Servan-Schreiber, D. & McClelland, J. L. (1988). Finite state automata and simple recurrent networks. *Neural Computation*, *1*, 372–381.

Cornet, J., Seron, X., Deloche, G. & Lories, G. (1988). Cognitive models of simple mental arithmetic: a critical review. *Cahiers de Psychologie Cognitive*, *8*(6), 551–571.

Cottrell, G. W. & Tsung, F. (1989). Learning simple arithmetic procedures. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 58–65. Lawrence Erlbaum Associates, Hillsdale, NJ.

Cox, L. (1974). Analysis, classification, and frequency of systematic error computational patterns in the addition, subtraction, multiplication, and division vertical algorithms for grades 2–6 and special education classes. Technical report ED 092 407, Kansas University Medical Center, Kansas City, KA.

Cummins, F. (1993). Representation of temporal patterns in recurrent networks. To appear in *The Fifteenth Annual Conference of the Cognitive Science Society*.

Dagenbach, D. & McCloskey, M. (1992). The organization of arithmetic facts in memory: evidence from a brain-damaged patient. Draft manuscript.

Dallaway, R. Z. (1992a). Memory for multiplication facts. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pp. 558–563. Lawrence Erlbaum Associates, Hillsdale, NJ.

Dallaway, R. Z. (1992b). The speed and slips of mental arithmetic. In Aleksander, I. & Taylor, J., eds, *Artificial Neural Networks II: Proceedings of the International Conference on Artificial Neural Networks*, pp. 1369–1372. Elsevier Science Publishers B.V.

Dennett, D. (1984). Cognitive wheels: the frame problem of AI. In Hookway, C., ed, *Minds, Machines and Evolution*. Cambridge University Press, Cambridge, UK.

Dennis, S. & Phillips, S. (1991). Analysis tools for neural networks. Technical report, University of Queensland, Australia.

Donaldson, M. (1978). *Children's Minds*. Fontana Press, London, UK.

Elman, J. L. (1988). Finding structure in time. Technical report CRL-8801, Center for Research in Language, University of California, San Diego, La Jolla, CA.

Elman, J. L. (1989a). Representation and structure in connectionist models. Technical report CRL 8903, Center for Research in Language, University of California, San Diego, La Jolla, CA.

Elman, J. L. (1989b). Structured representations and connectionist models. In *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, pp. 17–25. Lawrence Erlbaum Associates, Hillsdale, NJ.

Elman, J. L. (1991). Incremental learning, or The importance of starting small. Technical report CRL 9101, Center for Research in Language, University of California, San Diego, La Jolla, CA.

Fahlman, S. E. (1988). An empirical study of learning speed in back-propagation networks. Technical report CMU-CS-88-162, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Fahlman, S. E. (1991). The recurrent cascade-correlation architecture. Technical report CMU-CS-91-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Fahlman, S. E. & Lebiere, C. (1990). The cascade-correlation learning architecture. Technical report CMU-CS-90-100, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Giles, C. L., Miller, C. B., Chen, D., Chen, H. H., Sun, G. Z. & Lee, Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. Accepted for publication in *Neural Computation.*

Graham, D. J. (1990). Connectionist network models of cognitive arithmetic: explorations in back-propagation architectures and training sequences. Technical report AIM-8-90-3, Artificial Intelligence Laboratory, Department of Computer Science, University of Otago, New Zealand.

Groen, G. J. & Parkman, J. R. (1972). A chronometric analysis of simple addition. *Psychological Review*, *79*(4), 329–343.

Harley, W. S. (1991). *Associative Memory in Mental Arithmetic*. Ph.D. thesis, Johns Hopkins University, Baltimore, MD.

Hendler, J. A. (1989). Editorial: on the need for hybrid systems. *Connection Science*, *1*(3), 227–229.

Hennessy, S. (1990). Why bugs are not enough. In Elsom-Cook, M., ed, *Guided Discovery Tutoring: A Framework for ICAI Research*, chapter 10. Paul Chapman Publishing, London.

Hertz, J., Krogh, A. & Palmer, R. G. (1991). *Introduction to the Theory of Neural Computation*. Santa Fe Institute studies in the Sciences of Complexity. Addison-Wesley, Redwood City, CA.

Hinton, G. E. & Sejnowski, T. J. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. E., McClelland, J. L. & The PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations, chapter 7. MIT Press, Cambridge, MA.

Howell, A., Walker, R. & Fletcher, H. (1979). *Mathematics for Schools* (second edition), Vol. I and II of Teacher's resource book. Addison-Wesley, London.

Hughes, M. (1983). What is difficult about learning arithmetic?. In Donaldson, M., Grieve, R. & Pratt, C., eds, *Early Childhood Development and Education*, chapter 15. Basil Blackwell Ltd, Oxford, UK.

Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eight Annual Conference of the Cognitive Science Society*, pp. 531–545.

Kirsh, D. (1990). When is information explicitly represented?. In Hanson, P., ed, *Information, Thought and Content*. UBC Press.

Klahr, D. (1992). Information-processing approaches. In Vasta, R., ed, *Six Theories of Child Development*. Jessica Kingsley Publishers, London. First published in *Annals of Child Development*, volume 6, pp. 133–185.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Natural Selection*. MIT Press, Cambridge, MA.

Krueger, L. E. (1986). Why $2 \times 2 = 5$ looks so wrong: on the odd-even rule in product verification. *Memory and Cognition*, *14*(2), 141–149.

Laird, J. E., Newell, A. & Rosenbloom, P. S. (1986). Soar: an architecure for general intelligence. Technical report STAN-CS-86-1140, Department of Computer Science, Stanford University, Stanford, CA.

Marchman, V. A. (1992). Language learning in children and neural networks: plasticity, capacity and the critical period. Technical report 9201, Center for Research in Language, University of California, San Diego, La Jolla, CA.

Mareschal, D. & Shultz, T. R. (1993). A connectionist model of the development of seriation. To appear in the Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society.

Marr, D. (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. W. H. Freeman and Company, San Francisco, CA.

Maskara, A. & Noetzel, A. (1992). Forcing simple recurrent neural networks to encode context. To appear in *The Proceedings of the 1992 Long Island Conference on Artificial Intelligence and Computer Graphics.*

McClelland, J. L. (1979). On the time relations of mental processes: an examination of systems of processes in cascade. *Psychological Review*, *86*(4), 287–330.

McClelland, J. L. (1990). Parallel distributed processing: implications for cognition and development. In Morris, R. G. M., ed, *Parallel Distributed Processing: Implications for Psychology and Neurobiology*. Clarendon Press, Oxford.

McClelland, J. L. & Rumelhart, D. E. (1988). *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*. MIT Press, Cambridge, MA.

McCloskey, M., Aliminosa, D. & Sokol, S. M. (1991). Facts, rules, and procedures in normal calculation: evidence from multiple single-patient studies of impared arithmetic fact retrieval. *Brain and Cognition*, *17*, 154–203.

McCloskey, M. & Caramazza, A. (1985). Cognitive mechanisms in number processing and calculation: evidence from dyscalculia. *Brain and Cognition*, *4*, 171–196.

McCloskey, M. & Cohen, N. J. (1989). Catastrophic interference in connectionist networks: the sequential learning problem. In Bower, G. H., ed, *The Psychology of Learning and Motivation*, Vol. 24, pp. 109–165. Academic Press, New York.

McCloskey, M., Harley, W. & Sokol, S. M. (1991). Models of arithmetic fact retrieval: an evaluation in light of findings from normal and brain-damaged subjects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*(3), 377–397.

McCloskey, M. & Lindermann, A. M. (1992). MATHNET: Preliminary results from a distributed model of arithmetic fact retrieval. In Campbell, J. I. D., ed, *The Nature and Origins of Mathematical Skills*. Elsevier Science Publishers B.V.

McRae, K. & Hetherington, P. A. (1993). Catastrophic interference is eliminated in pre-trained networks. To appear in *Proceedings of the Fifteenth Conference of the Cognitive Science Society*.

Miller, K., Permutter, M. & Keating, D. (1984). Cognitive arithmetic: comparison of operations. *Journal of Experimental Psychology*, *10*(1), 46–60.

Mitchell, T. (1977). Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Vol. 1, pp. 305–310. Morgan Kaufmann Publishers.

Neches, R., Langley, P. & Klahr, D. (1987). Learning, development, and production systems. In Klahr, D., Langley, P. & Neches, R., eds, *Production Systems Models of Learning and Development*, chapter 1, pp. 1–53. MIT Press, Cambridge, MA.

Norman, D. A. (1988). *The Psychology of Everyday Things*. Basic Books, New York, NY.

Ohlsson, S. & Rees, E. (1991). The function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction*, *8*(2).

Payne, S. J. & Squibb, H. R. (1990). Algebra mal-rules and cognitive accounts of error. *Cognitive Science*, *14*, 445–481.

Pirolli, P. (1991). Book review: Mind bugs. *Artificial Intelligence*, *52*, 329–340.

Plunkett, K. & Marchman, V. A. (1990). From rote learning to system building. Technical report 9020, Center for Research in Language, University of California, San Diego, La Jolla, CA.

Plunkett, K. & Sinha, C. (1991). Connectionism and developmental theory. *Psykologisk Skriftserie Aarhus*, *16*(1).

Pollack, J. B. (1989a). Implications of recursive distributed representations. In Touretzkey, D. S., ed, *Advances in Neural Information Processing Systems*, Vol. 1. Morgan Kaufmann Publishers, San Mateo, CA.

Pollack, J. B. (1989b). Recursive distributed representations. Technical report, Ohio State University. Published in *Artificial Intelligence*, *46*(1–2):77–106.

Pylyshyn, Z. W. (1984). *Computation and Cognition: Towards a Foundation for Cognitive Science*. MIT Press, Cambridge, MA.

Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological Review*, *97*(2), 285–308.

Resnick, L. B. & Ford, W. W. (1982). *The Psychology of Mathematics for Instruction*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Rickard, T. C., Mozer, M. C. & Bourne, Jr., L. E. (1992). An interactive activation model of arithmetic fact retrieval. Technical report 92-15, Institute of Cognitive Science, University of Colorado, Boulder, CO.

Rose, D. E. (1991). Appropriate uses of hybrid systems. In Touretzky, D. S., Elman, J. L., Sejnowski, T. & Hinton, G., eds, *Proceedings of the 1990 Connectionist Models Summer School*, pp. 265–276. Morgan Kaufmann Publishers, San Mateo, CA.

Rosenbloom, P. S. (1989). A symbolic goal-oriented perspective on connectionism and soar. In Pfeifer, R., Schreter, Z., Fogelman-Soulié, F. & Steels, L., eds, *Connectionism in Perspective*. Elsevier Science Publishers B.V.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, D. E., McClelland, J. L. & The PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. MIT Press, Cambridge, MA.

Rumelhart, D. E. & McClelland, J. L. (1986a). On learning the past tenses of English verbs. In Rumelhart, D. E., McClelland, J. L. & The PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 2: Psychological and biological models. MIT Press, Cambridge, MA.

Rumelhart, D. E. & McClelland, J. L. (1986b). PDP models and general issues in cognitive science. In Rumelhart, D. E., McClelland, J. L. & The PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. MIT Press, Cambridge, MA.

Rumelhart, D. E., Smolensky, P., McClelland, J. L. & Hinton, G. E. (1986). Schemata and sequential thought processes in PDP models. In Rumelhart, D. E. & McClelland, J. L., eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 2: Psychological and biological models. MIT Press, Cambridge, MA.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L. (1988). Encoding sequential structure in simple recurrent networks. Technical report CMU-CS-88-183, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Servan-Schreiber, D., Cleeremans, A. & McClelland, J. L. (1991). Graded state machines: the representation of temporal contingencies in simple recurrent networks. *Machine Learning*, *7*, 161–193.

Shultz, T. R. (1991). Simulating stages of human cognitive development with connectionist models. In Birnbaum, L. & Collins, G., eds, *Machine Learning: Proceedings of the Eighth International Workshop*, pp. 105–109. Morgan Kaufmann Publishers, San Mateo, CA.

Shultz, T. R. & Schmidt, W. C. (1991). A cascade-correlation model of balance scale phenomena. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pp. 635–640. Lawrence Erlbaum Associates, Hillsdale, NJ.

Siegler, R. S. (1987). Strategy choices in subtraction. In Sloboda, J. A. & Rogers, D., eds, *Cognitive Processes in Mathematics*, pp. 81–106. Clarendon Press, Oxford.

Siegler, R. S. (1988). Strategy choice procedures and the development of multiplication skill. *Journal of Experimental Psychology: General*, *117*(3), 258–275.

Siegler, R. S. & Shrager, J. (1984). Strategy choices in addition and subtraction: how do children know what to do?. In Sophian, C., ed, *Origins of Cognitive Skills: The Eighteenth Annual Carnegie Symposium of Cognition*, pp. 229–293. Lawrence Erlbaum Associates, Hillsdale, NJ.

Sleeman, D. H. (1982). Assessing aspects of competence in basic algebra. In Sleeman, D. H. & Brown, J. S., eds, *Intelligent Tutoring Systems*, chapter 9. Academic Press, London.

Sloman, A. (1985). Why we need many knowledge representation formalisms. Technical report CSRP 52, School of Cognitive and Computing Sciences, University of Sussex, Brighton, UK.

Smolensky, P. (1986). Information processing in dynamic systems: foundations of harmony theory. In Rumelhart, D. E., McClelland, J. L. & The PDP Research Group, eds, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1: Foundations. MIT Press, Cambridge, MA.

Sokol, S. M., McCloskey, M., Cohen, N. J. & Aliminosa, D. (1991). Cognitive representations and processes in arithmetic: inferences from the performance of brain-damaged subjects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *17*(3), 355–376.

Stark, R. J. (1992). A symbolic/subsymbolic interface for variable instantiation. In Aleksander, I. & Taylor, J., eds, *Artificial Neural Networks II: Proceedings of the International Conference on Artificial Neural Networks*, pp. 753–756. Elsevier Science Publishers B.V.

Stazyk, E. H., Ashcraft, M. H. & Hamann, M. S. (1982). A network approach to mental multiplication. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *8*(4), 320–335.

Suppes, P. (1990). Eye-movement models for arithmetic and reading performance. In Kowler, E., ed, *Eye Movements and their Role in Visual and Cognitive Processes*, chapter 10. Elsevier Science Publishers B.V.

Suppes, P., Cohen, M., Laddaga, R., Anliker, J. & Floyd, R. (1983). A procedural theory of eye movements in doing arithmetic. *Journal of Mathematical Psychology*, *27*, 341–369.

Thornton, C. J. (1991). Stirring shakes: introduction to the special issue on hybrid systems. *Artificial Intelligence and the Simulation of Behaviour Quarterly*, *78*, 7.

Thornton, C. J. (1992a). Hybridism: a paradigm too many? (Introduction to part II of the special issue on hybrid models). *Artificial Intelligence and the Simulation of Behaviour Quarterly*, *79*, 9.

Thornton, C. J. (1992b). *Techniques in Computational Learning: An Introduction*. Chapman and Hall, London.

Touretzky, D. S. & Hinton, G. E. (1988). A distributed connectionist production system. *Cognitive Science*, *12*, 423–466.

van Gelder, T. (1992). What might cognition be if not computation?. Technical report 75, Cognitive Science Program, Indiana University, Bloomington, IN.

VanLehn, K. (1981). Bugs are not enough: empirical studies of bugs, impasses and repairs in procedural skills. Technical report CIS-11, Cognitive and Instructional Sciences Group, Xerox Palo Alto Research Center, Palo Alto, CA.

VanLehn, K. (1983). Felicity conditions for human skill acquisition: validating an AI-based theory. Technical report CIS-21, Cognitive and Instructional Sciences Group, Xerox Palo Alto Research Center, Palo Alto, CA.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, *31*, 1–40.

VanLehn, K. (1988). Towards a theory of impasse-driven learning. In Mandi, H. & Lesgold, A., eds, *Learning Issues for Intelligence Tutoring Systems*, chapter 2, pp. 19–41. Springer-Verlag, New York, NY.

VanLehn, K. (1990). *Mind Bugs: The Origins of Procedural Misconceptions*. MIT Press, Cambridge, MA.

VanLehn, K. (1991a). Rule acquisition events in the discovery of problem-solving strategies. *Cognitive Science*, *15*, 1–47.

VanLehn, K. (1991b). Two pseudo-students: applications of machine learning to formative evaluation. In Lewis, R. & Otsuki, S., eds, *Advanced Research on Computers in Education*, pp. 17–26. Elsevier Science Publishers.

Viscuso, S. R. (1989). *Memory for Arithmetic Facts: A Perspective Gained from Two Methodologies*. Ph.D. thesis, Department of Psychology, Brown University, Providence, RI.

Viscuso, S. R., Anderson, J. A. & Spoehr, K. T. (1989). Representing simple arithmetic in neural networks. In Tiberghien, G., ed, *Advances in Cognitive Science*, Vol. 2. Ellis Horwood, Chichester, UK.

Williams, R. J. & Zipser, D. (1989). Experimental analysis of the real-time recurrent learning algorithm. *Connection Science*, *1*(1), 87–111.

Winkerlman, J. H. & Schmidt, J. (1974). Associative confusions in mental arithmetic. *Journal of Experimental Psychology*, *102*(4), 734–736.

Young, R. M. (1974). Production systems as models of cognitive development. In *Proceedings of the AISB Summer Conference*.

Young, R. M. & O'Shea, T. (1981). Errors in children's subtraction. *Cognitive Science*, *5*, 153–177.

Zipser, D. (1990). Subgrouping reduces complexity and speeds up learning in recurrent networks. In Touretzky, D. S., ed, *Advances in Neural Information Processing Systems*, Vol. 1, pp. 638–641 San Mateo, CA. Morgan Kaufmann Publishers.

APPENDIX A

Addition Bugs

In comparison to the research on subtraction, there have been few studies which attempt to pin down multiplication and addition bugs. This lack of data is an obvious problem, although as pointed out in chapter 5, it is not such a serious problem for this particular project. Nevertheless, it is important to know, at least roughly, what kinds of bugs can occur and how frequent they are likely to be.

These appendices list the multiplication and addition bugs found in a search of the literature. All of the studies looked at have problems, and the lists presented here do not remove the need for a large, modern study of multiplication and addition.

A.1 Sources

The first source of bugs comes from Attisha (1983, also Attisha & Yazdani 1984). Attisha built a tutoring system for all four operations, and surveyed the literature for bugs. Although a large number of bugs were listed, Attisha failed to indicate their frequency or give examples with working marks. Without working marks it is difficult to interpret the definitions of certain bugs, namely those which involve carrying. Those definitions that could be interpreted are included here, and working marks have been added where it aids understanding of the bug.

One of the sources used by Attisha was the bug catalogue produced by Cox (1974). She studied the literature on arithmetic bugs from 1900 to 1973, and also conducted a study of bugs found in 564 subjects in grades two to six. The study demanded that the

Addition		Multiplication	
Renaming	23	Concept	19
Concept	17	Partial product	13
Wrong operation	6	× after renaming	10
Place value	5	+ after renaming	7
		Renaming	6
		× by zero	6
		Wrong operation	4
		Reversal of digits	2

Table A.1. Categories of bugs reported by Cox (1974), and the number of different bugs that fell under each category heading.

subjects be close to 100 per cent accurate on their number facts, and bugs were only accepted if they occurred at least three out of five times on a given type of problem.

Cox tested children on the four multicolumn tasks by having them complete test which were based around a number of levels. For addition there were eight levels, starting from addition of two digits to one digit without renaming. Each level became increasingly difficult, up to addition of three two-digit numbers with renaming. For multiplication there were ten levels. Cox listed the bugs found at each level, and then over the levels produced a categorization of the bugs. As can be seen from table A.1, Cox classified the bugs according to the kind of faulty knowledge that caused the error. The meaning behind Cox's labels is obvious, except for "concept", which is the case when the child seem to be lacking a basic understanding of the concept of multiplication or addition.

Cox's analysis and method of testing makes it impossible to know exactly how general or specific a particular bug is. For example, for level 2 problems (adding a one digit number to a two digit number, renaming needed) if the subject does not carry, is he or she exhibiting the bug does-not-carry-ones or does-not-carry? That is, if the subject was given a problem involving more digits, would they fail to carry just in the first (ones) column, or in all columns? In these appendices the most specific bugs are listed. Perhaps this complication is why Cox lists the global bug frequencies based on bug categories, rather than on specific bugs.

In her literature survey Cox missed the large arithmetic study undertaken by Buswell (1926). Using verbal protocols, Buswell described behaviours for all four operations. The study lists the number of occurrences of various “habits”—consistent behaviours, but not necessarily behaviours that could be classed as “buggy”. For example, one particular habit, “added carried number last”, describes a subject who always added the carry digit *after* adding up a column, rather than *before* starting on the column. Buswell notes that occasionally the subject would forget to add the carry, and this could be avoided if the subject added the carry first (ibid., p. 160). This kind of behaviour does not fit with the modern notion of a bug. However, many of the habits described do appear to be bugs, and are included here. Whereas Cox’s descriptions may be over specific, Buswell suffers the opposite problem. For example, Buswell describes the general bug wrong-operator, when other authors give more specific bugs like multiplies-instead-of-adding or subtracts-instead-of-adding.

The final source of bugs comes from a small, unpublished undergraduate project (Ainsworth 1991). It is included here because it builds upon the work of Young & O’Shea (1981) and presents a production system model of multiplication, as well as bug frequency information. As such it is the only study to date which can be easily compared to the computational studies of subtraction.

A.2 Notes on the catalogue entries

These two appendices list 102 bugs. There are 63 multiplication bugs, of which 9 do not have frequency information. For addition, the total is 39, 11 without frequency data.

Each entry in the catalogue is laid out as follows. The bug name, given in bold, is followed by a short description of the bug. Most bugs have one or more examples to clarify the description. The source of the bug is indicated by showing the name of one of the authors mentioned in the previous section. If that author gave frequency information, it is shown as a percentage of all the bugs taken from that author. Note that this will not be a percentage of all the bugs listed by that author: bugs were not included either because they were not clearly described, or because they were not relevant (e.g., number fact errors). For addition, 116 bug occurrences were used from Cox, and 484 from Buswell. A total of 113 occurrences of multiplication bugs were used from Cox, 512 from Buswell, and 76 from Ainsworth.

The catalogue is listed in alphabetical order, but table A.2 (on page 188) lists the most frequent bugs in order of frequency—or an approximation to that given that many bugs have frequency values from two or three authors.

Some of the bugs listed produce results that look identical to other bugs. For example, the bugs does-not-rename-sum and does-not-rename-product both result in the subject writing carry digits in the answer row. However, these bugs qualify as separate bugs because they have been observed independently of each other. That is, a subject can fail to rename a partial product, yet correctly rename when adding the partial product.

The Buswell frequencies are based on the total frequencies made by 263 subjects, spread over grades 3 to 6 (1926, tables XXXV to XXXVII, pp. 136–139). From the Ainsworth study, the frequencies are summed over two sets of 10–11 year olds and one group of 8-9 year olds (Ainsworth 1991, table 2, p. 32).

The model described in chapter 5 does not attempt to model certain kinds of errors, and for this reason some space-saving liberties have been taken in these appendices. In particular, pattern errors, like N×0=N, are only given one way round (i.e., 0×N=N is not shown). In the Cox, Ainsworth and Buswell studies, they are given both ways as they can occur independently.

Addition bugs

Added-imaginary-column. The subject went on to write an answer for a column that did not exist.

$$\begin{array}{rrrr} & & 6 & 9 \\ + & & 1 & 2 \\ \hline & 1 & 8_1 & 1 \end{array}$$

Buswell	0.21%

Adds-disregarding-columns. All the digits of the problem are added, without regard for the columns, i.e., 4+7+6+1+7=25.

$$\begin{array}{rrrr} & 4 & 7 & 6 \\ + & & 1 & 7 \\ \hline & & 2 & 5 \end{array}$$

Cox	12.93%
Buswell	11.36%
Attisha	

Adds-left-to-right. Addition is done horizontally, left to right. E.g., 2+4=6, 5+3=8.

$$\begin{array}{rrr} & 2 & 4 \\ + & 5 & 3 \\ \hline & 8 & 6 \end{array}$$

Buswell	0.62%
Attisha	

Adds-like-multiplication. Addition is performed using the pattern for multiplication (e.g., 3+6=9, 3+4=7).

$$\begin{array}{rccc} & & 4 & 6 \\ + & & & 3 \\ \hline & & 7 & 9 \end{array}$$

Cox 3.45%

Carries-one-to-100s. One is carried into the hundreds column regardless of whether a carry is or is not needed.

$$\begin{array}{rccc} & 5 & 0 & 5 \\ + & & 7 & 4 \\ \hline & 6_1 & 7 & 9 \end{array}$$

Cox 0.86%

Carries-one-to-10s. One is carried into the tens column when it is not necessary.

$$\begin{array}{rcc} & 4 & 6 \\ + & & 3 \\ \hline & 5_1 & 9 \end{array}$$

Cox 1.72%

Carries-ten. Ten is carried rather than one. I.e., 7+5=12, 2+1+10=13.

$$\begin{array}{rcc} & 2 & 5 \\ + & 1 & 7 \\ \hline 1 & 3 & 2 \end{array}$$

Attisha

Carries-two. The subject carries two in every column.

$$\begin{array}{rccc} & 2 & 7 & 1 \\ + & 4 & 1 & 2 \\ \hline & 8_2 & 0_2 & 3 \end{array}$$

Attisha

Carries-wrong-digit. When a column result needs to be carried, the wrong digit is carried.

$$\begin{array}{rccc} & 5 & 1 & 9 \\ + & & 8 & 6 \\ \hline & 9_4 & 1_5 & 1 \end{array}$$

Buswell 17.98%
Attisha

Carry-added-to-column. The carry digit is added into the answer for the current column. In this example, 1+3=4, 7+8=15, 1+5=6, 2+5=7.

$$\begin{array}{rccc} & 2 & 7 & 1 \\ + & 5 & 8 & 3 \\ \hline & 7 & 6 & 4 \end{array}$$

Cox 1.72%
Attisha

Carry-once-always-carry. Once the subject starts to carry a digit, it is always carried.

$$\begin{array}{rccc} & 1 & 2 & 7 \\ + & 4 & 5 & 6 \\ \hline & 6_1 & 8_1 & 3 \end{array}$$

Attisha

Carry-zero-units. When renaming, the carry digit is correctly noted, but the subject writes zero in the answer cell.

$$\begin{array}{rcc} & 7 & 5 \\ + & 1 & 8 \\ \hline & 9_1 & 0 \end{array}$$

Attisha

Column-skipped. One column is ignored and the column's answer is left blank.

$$\begin{array}{cccc} & 3 & 7 & 5 \\ + & 2 & 1 & 2 \\ \hline & 5 & & 7 \end{array}$$

Buswell 7.44%

Copies-first-addend. Where there is a single digit addend, that digit is copied as the answer in the ones column. The answer in the tens column is selected from one of the digits in the top row.

$$\begin{array}{ccc} & 4 & 6 \\ + & & 3 \\ \hline & 4 & 3 \end{array}$$

Cox 0.86%

Copies-ones-and-increments. The ones addend in the second row is incremented and given as the answer to the ones column.

$$\begin{array}{cccc} & 4 & 7 & 6 \\ + & & 1 & 7 \\ \hline & 4 & 8 & 8 \end{array}$$

Cox 0.86%

Copy-addend. One of the addend rows is copied to the answer row, possibly incremented or decremented.

$$\begin{array}{ccc} & 3 & 7 \\ + & 5 & 1 \\ \hline & 3 & 8 \end{array}$$

Cox 0.86%

Copy-lower-addend. The addend in the second row is copied to the answer. If there are digits over empty cells, they are also copied to the answer row.

$$\begin{array}{cccc} & 4 & 7 & 6 \\ + & & 1 & 7 \\ \hline & 4 & 1 & 7 \end{array}$$

Cox 0.86%

Does-not-carry. The subject does not carry.

$$\begin{array}{cccc} & 3 & 4 & 5 \\ + & & 7 & 6 \\ \hline & 3 & 1 & 1 \end{array}$$

Buswell 26.03%
Cox 1.72%
Attisha

Does-not-carry-ones. If the result of the ones column is a two digit number, the tens are not carried. The rest of the addition is correct

$$\begin{array}{cccc} & 3 & 4 & 5 \\ + & & 7 & 6 \\ \hline & 4_1 & 1 & 1 \end{array}$$

Cox 35.34%

Does-not-carry-over-blank. The subject does not carry to a number which is over an empty cell.

$$\begin{array}{cccc} & 4 & 6 & 8 \\ + & & & 9 \\ \hline & 4 & 6 & 7 \end{array}$$

Attisha

Does-not-raise-carry. The final carry at the end of an answer row is not raised onto the answer row.

$$\begin{array}{rrr} & 7 & 8 \\ + & 7 & 1 \\ \hline {}_1 & 4 & 9 \end{array}$$

Buswell 7.02%

Does-not-record-100s. The hundreds column answer is not recorded on the answer row.

$$\begin{array}{rrrr} & 5 & 0 & 5 \\ + & & 7 & 4 \\ \hline & 0 & 7 & 9 \end{array} \qquad \begin{array}{rrrr} & 4 & 7 & 6 \\ + & & 1 & 7 \\ \hline & & 9_1 & 3 \end{array}$$

Cox 1.72%

Does-not-rename-copy-100s. The sum of the first column is not renamed, the tens column is not processed, and the digit in the hundreds column is copied to the answer row.

$$\begin{array}{rrrr} & 2 & 0 & 5 \\ + & & 8 & 6 \\ \hline & 2 & 1 & 1 \end{array}$$

Cox 0.86%

Does-not-rename-quits-100s. The carry digit from the first addition is written in the answer row but the hundreds column is not processed.

$$\begin{array}{rrrr} & 2 & 0 & 5 \\ + & & 8 & 6 \\ \hline & 8 & 1 & 1 \end{array}$$

Cox 1.72%

Does-not-rename-sum. During addition, digits to be carried are written on the answer row.

$$\begin{array}{rrr} & 4 & 8 \\ + & & 3 \\ \hline 4 & 1 & 1 \end{array} \qquad \begin{array}{rrrr} & & 2 & 8 \\ & \times & 1 & 7 \\ \hline & 1_1 & 9_5 & 6 \\ + & 2 & 8 & 0 \\ \hline 3 & 1 & 7 & 6 \end{array}$$

Cox	18.97%
Buswell	3.1%
Attisha	

Ignores-10s-column. The tens column is ignored.

$$\begin{array}{rrr} & 4 & 8 \\ + & & 3 \\ \hline & 1 & 1 \end{array}$$

Cox 0.86%

Ignores-first-column. The first column of the problem is ignored.

$$\begin{array}{rrrr} & 3 & 2 & 5 \\ + & 2 & 7 & 1 \\ \hline & 5 & 9 & \end{array}$$

Attisha

Left-alignment. The subject writes the problem aligned against the left column.

$$\begin{array}{rrr} & 5 & 4 \\ + & 3 & \\ \hline & 8 & 4 \end{array}$$

Attisha

Multiplies-instead-of-adding. The subject multiplies, rather than adding.

$$\begin{array}{rrr} & 3 & 4 \\ + & & 2 \\ \hline & 6 & 8 \end{array}$$

Attisha

N+N=N. The subject answers that the sum of two identical digits is just one of the digits.

$$\begin{array}{rrr} & 3 & 2 \\ + & 4 & 2 \\ \hline & 7 & 2 \end{array}$$

Attisha

One-one-too-many. The answer in the ones column is one more than it should be.

$$\begin{array}{rrr} & 5 & 2 \\ & 8 & 6 \\ + & 1 & 4 \\ \hline 1 & 5 & 3 \end{array}$$

Cox 1.29%

Quit-after-last-lower. When the last of the numbers in the lower row has been processed, the subject quits.

$$\begin{array}{rrrr} & 2 & 7 & 3 \\ + & & 2 & 4 \\ \hline & & 9 & 7 \end{array}$$

Attisha

Quits-when-carry. When a carry is needed, the subject quits.

$$\begin{array}{rrrr} & 2 & 7 & 3 \\ + & 1 & 8 & 2 \\ \hline & & & 5 \end{array}$$

Attisha

Renames-to-wrong-column. When renaming, the subject renames the carry to the wrong column. In the example, the carry from the units column was renamed to the hundreds column.

$$\begin{array}{rrrr} & 4 & 7 & 6 \\ + & & 1 & 7 \\ \hline & 5_1 & 8 & 3 \end{array}$$

Cox 3.88%

Spurious-carry. As some stage in the sum a carry was added when it was not appropriate.

Buswell 5.99%

Stutter-add. When there is an empty cell in the problem, the last digit in the bottom row is used as the addend.

$$\begin{array}{rrrr} & 4 & 2 & 1 \\ + & & 3 & 4 \\ \hline & 7 & 5 & 5 \end{array}$$

Buswell 3.72%
Attisha

Subtract-carry. The carry was subtacted, rather than added to the addend.

$$\begin{array}{rrr} & & 7 \\ + & 8 & 9 \\ \hline & 7_1 & 6 \end{array}$$

Buswell 0.21%

Subtracts-instead-of-adding. The subject uses the subtraction algorithm instead of addition algorithm.

$$\begin{array}{rr} & 15 \\ + & 2 \\ \hline & 13 \end{array}$$

Cox	9.48%
Attisha	

Wrong-operator. As some stage in the problem the wrong operator was used (e.g., multiplication for addition). Buswell was no more specific than this.

Buswell	16.32%

	Addition			Multiplication			
Rank	Bug	B	C	Bug	A	B	C
1	does-not-carry-ones		35.34	N×0=N	21.05	23.44	12.39
2	does-not-carry	26.03	1.72	answer-on-one-row	27.63		
3	adds-disregarding-columns	11.36	12.93	does-not-add-carry		17.38	4.42
4	does-not-rename-sum	3.1	18.97	carries-wrong-number		18.55	
5	carries-wrong-digit	17.98		multiplies-using-addition-pattern	11.84		4.42
6	wrong-operator	16.32		forgets-annex	3.95	7.62	3.54
7	subtracts-instead-of-adding		9.48	quits-after-first-multiplier	2.63	10.16	
8	column-skipped	7.44		adds-carry-and-multiplier			8.85
9	does-not-raise-carry	7.02		carry-added-to-multiplicand		0.78	7.96
10	spurious-carry	5.99		copies-after-first-column			7.96
11	renames-to-wrong-column		3.88	does-not-rename-product	6.58		0.88
12	stutter-add	3.72		carries-wrong-digit	3.95	1.76	0.88
13	adds-like-multiplication		3.45	partial-product-confusion		6.25	
14	carries-one-to-10s		1.72	incorrect-number-of-annex-zeros			6.19
15	carry-added-to-column		1.72	adds-multiplicand-to-answer		5.47	
16	does-not-record-100s		1.72	multiplies-carry			5.31
17	does-not-rename-quits-100s		1.72	copies-multiplicand		0.78	4.42
18	one-one-too-many		1.29	does-not-carry-to-10s	3.95		0.88
19	carries-one-to-100s		0.86	partial-product-incorrectly-summed			4.42
20	copies-first-addend		0.86	multiplies-all-by-first-multiplier	3.95		
21	copies-ones-and-increments		0.86	ignores-zero-multiplier			3.54
22	copy-addend		0.86	digit-omitted		3.32	
23	copy-lower-addend		0.86	adds-instead-of-multiplying			2.65
24	does-not-rename-copy-100s		0.86	no-annexing-in-third			2.65
25	ignores-10s-column		0.86	0×N=0-carry-N	2.63		
26	adds-left-to-right	0.62		multiplies-partial-product	2.63		
27	added-imaginary-column	0.21		does-not-add-partial-product		2.34	
28	subtract-carry	0.21		partial-product-reversed		1.17	0.88

Table A.2. The 28 most frequent addition and multiplication bugs. *Key:* Values are percentages from three authors, A=Ainsworth (1991), N=76; B=Buswell (1926), N=512 for multiplication, N=484 for addition; C=Cox (1974), N=113 for multiplication, N=116 for addition.

Appendix B

Multiplication Bugs

0×N=0-carry-N. When multiplying by zero, zero is written as the column's answer, but the multiplicand is carried.

$$\begin{array}{rrr} & 2 & 0 \\ \times & & 3 \\ \hline & 9_3 & 0 \end{array}$$

Ainsworth 2.63%

Adds-carry-and-multiplicand. The carried digit is added to the multiplicand, and this sum is given as the column answer. E.g., 6×8=48, 3+4=7. The final "5" was copied.

$$\begin{array}{rrrr} & 5 & 3 & 6 \\ \times & & & 8 \\ \hline & 5 & 7_4 & 8 \end{array}$$

Attisha

Adds-carry-and-multiplier. The carried digit is added to the multiplier, and this sum is given as the column answer. I.e., 4×5=20, 4+2=6, 4×8=32.

$$\begin{array}{rrrr} & 8 & 0 & 5 \\ \times & & & 4 \\ \hline 3 & 2 & 6_2 & 0 \end{array}$$

Cox 8.85%

Adds-carry-and-multiplier-when-zero. When the multiplicand is a zero, the subject adds the carry digit and the multiplier to obtain an answer. In the example, 2×7=14, 1+2=3, 2×5=10.

$$\begin{array}{rrrr} & 5 & 0 & 7 \\ \times & & & 2 \\ \hline 1 & 0 & 3_1 & 4 \end{array}$$

Cox 0.88%

Adds-carry-to-multiplicands. A column's answer is the sum of the carry digit and the multiplicand. E.g., 6×8=48, 3+4=7, 5+4=9.

$$\begin{array}{rrrr} & 5 & 3 & 6 \\ \times & & & 8 \\ \hline & 9 & 7_4 & 8 \end{array}$$

Attisha

Adds-carry-to-product. When the result of a multiplication is a two digit number, those numbers are added, e.g., 3×5=15=6.

$$\begin{array}{rrrr} & & 5 & 2 \\ & \times & 1 & 3 \\ \cline{2-4} & & 6 & 6 \\ + & 5 & 2 & 0 \\ \hline & 5 & 8 & 6 \end{array}$$

Ainsworth	1.32%

Adds-instead-of-multiplying. The addition algorithm is used instead of multiplication.

$$\begin{array}{rrrr} & 7 & 2 & 5 \\ \times & & & 3 \\ \hline & 7 & 2 & 8 \end{array}$$

Cox	2.65%
Attisha	

Adds-multiplicand-to-answer. A multiplicand is not multiplied, but instead is added to the answer. I.e., 3×6=18, 7+1=8.

$$\begin{array}{rrr} & 7 & 6 \\ \times & & 3 \\ \hline & 8_1 & 8 \end{array}$$

Buswell	5.47%

Adds-using-multiplication-pattern. The subject uses the pattern for multiplication, but adds the digits.

$$\begin{array}{rrrr} & 3 & 2 & 0 \\ \times & & & 4 \\ \hline & 7 & 6 & 4 \end{array}$$

Cox	1.77%
Attisha	

Always-carries. The subject always adds in the carry digit.

$$\begin{array}{rrrr} 2 & 4 & 2 & 9 \\ & & & 2 \\ \hline 5 & 9 & 5_1 & 8 \end{array}$$

Buswell	0.2%
Attisha	

Always-carries-one. When a carry occurs, the subject adds one to a column answer, not the real carry.

$$\begin{array}{rrrr} & 5 & 1 & 4 \\ \times & & & 7 \\ \hline 3 & 5 & 8_2 & 8 \end{array}$$

Attisha

Answer-on-one-row. All the partial products are written on one answer row.

$$\begin{array}{rrrr} & & 2 & 3 \\ & \times & 4 & 8 \\ \cline{2-4} 9_1 & 3_1 & 8_2 & 4 \end{array}$$

Ainsworth	27.63%

Answers-left-to-right. The subject writes the answer left to right. In the example, 2×9=18, subject writes 8 carries 1, and so on.

$$\begin{array}{rrrr} & 7 & 1 & 2 \\ \times & & & 9 \\ \hline 8 & 0_1 & 6_1 & 4 \end{array}$$

Attisha

Carries-wrong-digit. When the result of a multiplication or addition is a number that needs to be carried, the wrong digit is carried.

$$\begin{array}{rrrr} & 7 & 2 & 4 \\ \times & & & 6 \\ \hline 4 & 8_6 & 1_4 & 2 \end{array}$$

Ainsworth	3.95%
Buswell	1.76%
Cox	0.88%
Attisha	

Carries-wrong-number. A composite bug, where some number was carried, but it was the wrong one (e.g., the units number as in carries-wrong-digit, or always a one, as in always-carries-one).

Buswell	18.55%

Carry-added-to-multiplicand. The carry digit is added to the multiplicand before multiplying. I.e., 6×7=42, (2+4)×6=36, (3+3)×6=36.

$$\begin{array}{rrrr} & 3 & 2 & 7 \\ \times & & & 6 \\ \hline 3 & 6_3 & 6_4 & 2 \end{array}$$

Cox	7.96%
Buswell	0.78%
Attisha	

Carry-added-to-tens. When adding a carry digit to a product, the carry is added to the tens part, e.g., 4×6=24, 4×2=8, 2+8=28.

$$\begin{array}{rrrr} & & 2 & 6 \\ & \times 1 & & 4 \\ \hline & 2 & 8_2 & 4 \\ + & 2 & 6 & 0 \\ \hline & 5_1 & 4 & 4 \end{array}$$

Ainsworth	1.32%

Carry-not-raised. The carry digit is not raised at the end of a answer row in the partial product.

$$\begin{array}{rrrr} & & 4 & 2 \\ & \times & 4 & 1 \\ \hline & & 4 & 2 \\ + & {}_16 & 8 & 0 \\ \hline & 7_1 & 2 & 2 \end{array}$$

Ainsworth	1.32%

Carry-once-always-carry. Once the subject starts to carry a digit, it is always carried.

$$\begin{array}{rrrr} & 1 & 1 & 2 \\ \times & & & 7 \\ \hline & 8_1 & 8_1 & 4 \end{array}$$

Cox	0.88%

Copies-after-first-column. The first column of a problem is solved correctly, but the remaining multiplicands are copied to the answer row.

$$\begin{array}{rrrr} & 3 & 1 & 3 \\ \times & & & 3 \\ \hline & 3 & 1 & 9 \end{array}$$

Cox	7.96%
Attisha	

Copies-multiplicand. No multiplication is performed, but the multiplicand is copied to the answer row.

$$\begin{array}{cccc} & 2 & 0 & 0 \\ \times & & & 4 \\ \hline & 2 & 0 & 0 \end{array}$$

Cox	4.42%
Buswell	0.78%
Attisha	

Copies-multiplicand-at-100s. When processing the hundreds multiplier, the subject inserts two zeros and copies the multiplicand.

$$\begin{array}{ccccc} & & 5 & 1 & 9 \\ & \times 4 & 0 & 2 \\ \cline{2-5} & 1 & 0 & 3 & 8 \\ 5 & 1 & 9 & 0 & 0 \end{array}$$

Cox	0.88%

Copies-multiplicand-including-zero. The multiplicand is copied as the answer, but a zero is first inserted into the answer.

$$\begin{array}{cccc} & 2 & 4 & 7 \\ \times & & 2 & 0 \\ \hline 2 & 4 & 7 & 0 \end{array}$$

Cox	0.88%

Copies-multiplicand-less-2. The answer is two less than the multiplicand.

$$\begin{array}{ccc} & 1 & 6 \\ \times & & 4 \\ \hline & 1 & 4 \end{array}$$

Cox	0.88%

Cross-multiplies. The digits of the problem are cross multiplied, e.g., 1×4=4, 3×2=6.

$$\begin{array}{ccc} & 4 & 2 \\ \times & 3 & 1 \\ \hline & 6 & 4 \end{array}$$

Ainsworth	1.32%

Digit-omitted. A digit in the product is not written down. In the example, the subject decided not to write down the 5 from 54 (8×8=64, 8×6=48 + 6 = 54).

$$\begin{array}{ccccc} & & & 6 & 8 \\ \times & 9 & 8 & 7 & 8 \\ \hline & & & 4_6 & 4 \end{array}$$

Buswell	3.32%

Does-not-add-carry. The carry digit is not added to the column product. Cox notes this error when the subject misses just one carry in a problem (not necessarily every carry).

$$\begin{array}{cccc} & 1 & 4 & 9 \\ \times & & & 4 \\ \hline & 4_1 & 6_3 & 6 \end{array}$$

Buswell	17.38%
Cox	4.42%
Attisha	

Does-not-add-partial-product. The subject does not add the partial product, leaving the sum as shown in the example.

$$\begin{array}{ccccc} & & & 5 & 3 \\ & \times 3 & 2 & 1 \\ \cline{2-5} & & & 5 & 3 \\ & 1 & 0 & 6 & 0 \\ 1 & 5 & 9 & 0 & 0 \end{array}$$

Buswell	2.34%

Does-not-carry-in-partial-product. The subject does not carry when adding the partial product.

$$\begin{array}{r} 9\ 2\ 7 \\ \times\quad 7\ 3 \\ \hline 2\ 7\ 8\ 1 \\ 6\ 4\ 8\ 9\ 0 \\ \hline 6\ 6\ 5\ 7\ 1 \end{array}$$

Attisha	

Does-not-carry-to-10s. The carried digit is not added to the product in the tens column.

$$\begin{array}{r} 2\ 1\ 6 \\ \times\qquad 6 \\ \hline 1\ 2\ 6_3\ 6 \end{array}$$

Ainsworth	3.95%
Cox	0.88%
Attisha	

Does-not-rename-copies-10s. The product from the first multiplication is written in the answer row without renaming, and the tens multiplicand is copied into the answer.

$$\begin{array}{r} 1\ 6 \\ \times\quad 4 \\ \hline 1\ 2\ 4 \end{array}$$

Cox	0.88%

Does-not-rename-first-then-copies. The first multiplication is performed, and the answer is written in the answer without renaming, and remaining multiplicands are copied.

$$\begin{array}{r} 2\ 3\ 7 \\ \times\qquad 4 \\ \hline 2\ 3\ 2\ 8 \end{array}$$

Attisha	

Does-not-rename-product. Digits carried over from a multiplication are written on the answer row.

$$\begin{array}{r} 1\ 7 \\ \times\quad 5 \\ \hline 5\ 3\ 5 \end{array}$$

Ainsworth	6.58%
Cox	0.88%

Forgets-annex. The zero is forgotten. In the example, a zero should have been inserted into the second answer row.

$$\begin{array}{r} 4\ 5 \\ \times\ 2\ 9 \\ \hline 4\ 0_4\ 5 \\ +\quad 9_1\ 0 \\ \hline 4\ 9\ 5 \end{array}$$

Buswell	7.62%
Ainsworth	3.95%
Cox	3.54%

Ignores-zero-multiplier. The first multiplier is ignored when it's a zero, and no zero is inserted in the answer row.

$$\begin{array}{r} 5\ 3 \\ \times\ 2\ 0 \\ \hline 1\ 0\ 6 \end{array}$$

Cox	3.54%
Attisha	

Incorrect-number-of-annex-zeros. An incorrect number of zeros are inserted into one of the answer rows.

$$\begin{array}{rrrrrr}
 & & 4 & 5 & 6 \\
 & \times & 2 & 5 & 1 \\ \cline{2-5}
 & & 4 & 5 & 6 \\
2 & 2 & 8 & 0 & 0 \\
9 & 1 & 2 & 0 & 0
\end{array}$$

Cox 6.19%

Last-digits-multiplied. The last multiplicand is multiplied by the last multiplier, rather than multiply each multiplier by each multiplicand. In the example, 2×7=14, 2×0=0+1=1, then 5×3=15.

$$\begin{array}{rrrr}
 & 5 & 0 & 7 \\
\times & & 3 & 2 \\ \hline
1 & 5 & 1_1 & 4
\end{array}$$

Cox 1.77%

Last-multiplication-skipped. The second multiplicand is not multiplied by the second multiplier.

$$\begin{array}{rrr}
 & 3 & 2 \\
\times & 4 & 1 \\ \hline
 & 3 & 2 \\
+ & 8 & 0 \\ \hline
1 & 1 & 2
\end{array}$$

Ainsworth 1.32%

Multiplied-product-by-carry. The carry digit is multiplied by the product, rather than being added to it. In this example, 3×9=27, 3×1=3, 3×2=6.

$$\begin{array}{rrr}
 & 1 & 9 \\
\times & & 3 \\ \hline
 & 6_2 & 7
\end{array}$$

Cox 1.77%
Attisha

Multiplies-all-by-first-multiplier. The first multiplier is used to multiply all the other digits. In this example, 1×2=2, 1×4=4, 1×3=3.

$$\begin{array}{rrr}
 & 4 & 2 \\
\times & 3 & 1 \\ \hline
3 & 4 & 2
\end{array}$$

Ainsworth 3.95%

Multiplies-by-carry-over-blank. When the multiplicand is over an empty cell, the subject multiplies by the carry digit.

$$\begin{array}{rrr}
 & 7 & 6 \\
\times & & 4 \\ \hline
1 & 4_2 & 4
\end{array}$$

Attisha

Multiplies-carry. When there is a carry digit in the current column, it is used for multiplication instead of the multiplicand. I.e., 8×4=32, 3×4=12, and so on.

$$\begin{array}{rrrr}
 & 3 & 0 & 8 \\
\times & & & 4 \\ \hline
 & 4_1 & 2_3 & 2
\end{array}$$

Cox 5.31%
Attisha

Multiplies-last-multiplicand-and-writes-10. The only multiplication performed is to multiply the multiplier by the last multiplicand (3×6 in the example). The product is written in the answer row, and ten is written after it.

$$\begin{array}{rrrr} & & 3 & 0 \\ & \times & & 6 \\ \hline 1 & 0 & 1 & 8 \end{array}$$

Cox	0.88%

Multiplies-multiplicands. The first multiplication is correct, but the subject then multiplies the multiplicands. In this example, 1×4=4, 2×4=8.

$$\begin{array}{rrr} & 2 & 4 \\ \times & 3 & 1 \\ \hline & 8 & 4 \end{array}$$

Ainsworth	1.32%

Multiplies-partial-product. The partial product is multiplied, not added, with the bug multiplies-using-addition-pattern.

$$\begin{array}{rrr} & 3 & 2 \\ \times & 2 & 1 \\ \hline & 3 & 2 \\ 6 & 4 & 0 \\ \hline 7_1 & 2 & 0 \end{array}$$

Ainsworth	2.63%

Multiplies-using-addition-pattern. Uses the addition pattern, but multiplies.

$$\begin{array}{rrrr} & 5 & 2 & 4 \\ \times & 7 & 3 & 1 \\ \hline 3 & 5 & 6 & 4 \end{array}$$

Ainsworth	11.84%
Cox	4.42%
Attisha	

Multiply-by-carry-when-zero. When the multiplicand is zero, the subject prefers to multiply by the carry digit.

$$\begin{array}{rrrrr} & & 4 & 0 & 6 \\ & \times & & 7 & 3 \\ \hline & 1 & 2 & 3_1 & 8 \\ 3 & 0_2 & 8_4 & 2 & 0 \end{array}$$

Cox	1.77%
Attisha	

N×0=N. When N is multiplied by zero, N is the answer.

$$\begin{array}{rrrr} & 3 & 0 & 2 \\ \times & & & 3 \\ \hline & 9 & 3 & 6 \end{array}$$

Buswell	23.44%
Ainsworth	21.05%
Cox	12.39%
Attisha	

No-annexing-in-third. No zeros were inserted for the third answer row.

Cox	2.65%

Partial-product-confusion. A general, combination error in which the subject had difficulty when the problem had two or more multipliers. In the first example, 4×5=20, 2×4=8+2=10, 2×1=2+1=3. In the second example, the second and third products are written on the same answer row.

$$\begin{array}{r} 1\;4\;4 \\ 2\;5 \\ \hline 3_1\;0_2\;0 \end{array} \qquad \begin{array}{r} 5\;1\;2 \\ \times \quad 2\;5 \\ \hline 5\;1\;2 \\ +\,1\;0\;2\;4\;1\;5\;3\;6 \\ \hline \end{array}$$

Buswell 6.25%

Partial-product-incorrectly-summed. The addition of the partial product is incorrect. Cox apparently used this category to cover a number of addition bugs.

$$\begin{array}{r} 5\;3 \\ \times\,7\;4 \\ \hline 2\;1\;2 \\ +\,3\;7\;1\;0 \\ \hline 4\;4\;2\;2 \end{array}$$

Cox 4.42%

Partial-product-reversed. The order of the digits is reversed in the partial product. In the example, the “219” should be “912”.

$$\begin{array}{r} 4\;5\;6 \\ \times\,2\;5\;1 \\ \hline 4\;5\;6 \\ 2\;2\;8\;0 \\ 2\;1\;9 \end{array}$$

Buswell 1.17%
Cox 0.88%

Quits-after-first-multiplication. Only the first multiplication is completed.

$$\begin{array}{r} 2\;4\;7 \\ \times \quad\;\; 4 \\ \hline 2\;8 \end{array}$$

Attisha

Quits-after-first-multiplier. Only the first multiplier is used.

$$\begin{array}{r} 3\;4\;6 \\ \times \quad 2\;8 \\ \hline 2\;\;7_3\,6_4\,8 \end{array}$$

Buswell 10.16%
Ainsworth 2.63%
Attisha

Quits-at-100s. The subject quits multiplying after processing the tens column.

$$\begin{array}{r} 2\;2\;4 \\ \times\,1\;1\;8 \\ \hline 1\;7\;9\;2 \\ 2\;2\;4\;0 \end{array}$$

Cox 0.88%

Repeated-multiplication. A multiplication was repeated.

$$\begin{array}{r} 4 \\ \times\,2 \\ \hline 8\;8 \end{array}$$

Buswell 0.59%

Skips-zero-multiplicand. When the multiplicand contains a zero, the multiplication is skipped and the reminding digits of the multiplicand are multiplied by the multiplier directly under the zero. In the example, 2×9=18, 8×5=40.

$$\begin{array}{cccc}
 & 8 & 0 & 9 \\
\times & & 5 & 2 \\
\hline
4 & 0 & 1 & 8
\end{array}$$

Attisha

Spurious-zero-in-100s. A zero is inserted in the hundreds column for no apparent reason.

$$\begin{array}{ccccc}
 & & 9 & 0 & 5 \\
 & \times & & 4 & 6 \\
\cline{2-5}
5 & 4 & 0 & 3 & 0 \\
3 & 6 & 0 & 2 & 0
\end{array}$$

Cox 0.88%

Subtracts-partial-product. The subject subtracts the partial product rather than adding. In this example the subject also subtracts the smaller number from the larger.

$$\begin{array}{cccc}
 & & 5 & 3 \\
 & \times & 7 & 4 \\
\cline{2-4}
 & 2 & 1 & 2 \\
3 & 7 & 1 & 0 \\
\hline
3 & 5 & 0 & 2
\end{array}$$

Cox 0.88%
Attisha

Too-many-annex-zeros. Too many zeros are inserted into the answer row when multiplying by a multiple of ten.

$$\begin{array}{ccccccc}
 & & & & 5 & 5 & 3 \\
 & & & \times & & 2 & 0 \\
\cline{4-7}
1 & 1 & 0 & 6 & 0 & 0 & 0
\end{array}$$

Cox 0.88%
Attisha

Weird-order. The digits are multiplied in a strange order. In this example, the order is: 4×1=4, 2×1=2, 2×3=6, 2×4=8.

$$\begin{array}{cccc}
 & & 1 & 3 \\
 & \times & 2 & 4 \\
\cline{2-4}
8 & 6 & 2 & 4
\end{array}$$

Ainsworth 1.32%

Works-left-to-right. The subject starts at the left, adding carries to the right. In the example, 5×3=15, 2×3=6+1=7.

$$\begin{array}{ccc}
 & 5 & 2 \\
\times & & 3 \\
\hline
 & 5 & 7_1
\end{array}$$

Buswell 0.2%

Zero-in-first-row. A zero is inserted at the start of the first row. Subsequent rows have the correct number of zeros.

$$\begin{array}{cccccc}
 & & & 4 & 3 & 6 \\
 & & \times & & 5 & 1 \\
\cline{3-6}
 & & 4 & 3 & 6 & 0 \\
+ & 2 & 1_1 & 8_3 & 0 & 0 \\
\hline
 & 2 & 6_1 & 1 & 6 & 0
\end{array}$$

Cox 0.88%
Attisha