COGNITIVE SCIENCE **12**, 423–466 (1988)

A Distributed Connectionist Production System

DAVID S. TOURETZKY
GEOFFREY E. HINTON
Carnegie Mellon University

DCPS is a connectionist production system interpreter that uses distributed representations. As a connectionist model it consists of many simple, richly interconnected neuron-like computing units that cooperate to solve problems in parallel. One motivation for constructing DCPS was to demonstrate that connectionist models are capable of representing and using explicit rules. A second motivation was to show how "coarse coding" or "distributed representations" can be used to construct a working memory that requires far fewer units than the number of different facts that can potentially be stored. The simulation we present is intended as a detailed demonstration of the feasibility of certain ideas and should not be viewed as a full implementation of production systems. Our current model only has a few of the many interesting emergent properties that we eventually hope to demonstrate: It is damage-resistant, it performs matching and variable binding by massively parallel constraint satisfaction, and the capacity of its working memory is dependent on the similarity of the items being stored.

1. INTRODUCTION

DCPS is a connectionist production system interpreter that uses distributed representations. As a connectionist model (Feldman & Ballard, 1982), it consists of many simple, richly interconnected neuron-like computing units that cooperate to solve problems in parallel. One motivation for constructing DCPS was to demonstrate that distributed connectionist models are capable of representing and using explicit rules. Earlier connectionist models (Rumelhart & McClelland, 1986) have shown that many phenomena which appear to require explicit rules can be handled by using connection strengths that implicitly capture the regularities of the task domain without ever making these regularities explicit. However, we do not believe that this removes the need for a more explicit representation of rules in tasks that

This work was sponsored by a grant from the System Development Foundation, and by National Science Foundation grants IST-8516330 and IST-8520359. We thank Scott Fahlman, Jay McClelland, David Rumelhart, and Terry Sejnowski for helpful discussions, Bruce Krulwich for contributing to the continued development of the simulation, and the anonymous referees for several valuable suggestions.

Correspondence and requests for reprints should be sent to David S. Touretzky, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213-3890.

more closely resemble serial, deliberate reasoning. A person can be told an explicit rule such as "i before e except after c" and can then apply this rule to the relevant cases. The person is aware of which rule is being applied and cannot apply a large number of different rules in parallel, though he or she may be able to perform a parallel search for the appropriate rule.

The natural way to implement explicit rules in a connectionist network is to apply a parallel best-fit search to the task of finding the rule whose left-hand side best matches the current contents of working memory. Connectionist networks are good at performing pattern matching, especially when there is no perfect match and the aim is to find the best partial match. One difficulty with this approach is that the kind of matching required to implement a production system is more complex than simple template matching. The left-hand side of a production may contain several instances of the same variable, and matches are only valid if all instances of the variable receive the same binding. Ensuring consistent variable bindings in a parallel network is a difficult and important problem (Barnden, 1984) and one of the main aims of this paper is to demonstrate a feasible solution. The need to bind variables consistently is not just an artifact of using explicit rules. To capture truly the regularities that are normally expressed by rules containing variables, an information-processing system must do something equivalent to variable binding, even if the implementation does not use explicit rules. Attempts to eliminate variables by expanding all their possible instantiations do not really capture the regularities, and they certainly do not allow correct generalizations from limited training data.

Ballard and Hayes have demonstrated that a rather elaborate connectionist network can decide whether two expressions can be unified (Ballard, 1986; Ballard & Hayes, 1984). DCPS uses a different solution which is based on earlier work (Hinton, 1981a) on viewpoint-invariant shape recognition. In matching an object model to a retinal image, it is essential to ensure that all the matches of a piece of the model to a piece of the image assume the same viewpoint. In matching the LHS of a rule to the contents of working memory, it is essential to ensure that all the matches of a clause in the LHS to a fact in working memory assume the same variable bindings.

A second motivation for DCPS is to show how "coarse coding" or "distributed representations" can be used to construct a working memory that requires far fewer units than the number of different facts that can potentially be stored. The price of this economy is that only a small fraction of the potential facts can actually be present in working memory at any one time. Earlier analyses of coarse coding have shown that it is efficient (Hinton, 1981b; Hinton, McClelland, & Rumelhart, 1986) but they have failed to demonstrate that it can be used effectively when many different groups of units must interact correctly. Coarse coding "smears" the representation of a given item across many units, and when coarse-coded representations in several different groups of units interact during an iterative best-fit

search, there is a danger that the representation will become progressively more smeared with each iteration.

The simulation we present is intended as a detailed demonstration of the feasibility of certain ideas and should not be viewed as a full implementation of production systems. The production rules our model interprets are much simpler than those found in OPS5 or EMYCIN. Nevertheless, they do contain variables that get bound consistently by the connectionist network, and they are implemented using distributed representations throughout. This falsifies any strong claim that connectionist systems using distributed representations could not possibly implement symbol processing. However, it leaves us upon to the alternative criticism that we have merely implemented a very simple production system in peculiarly inefficient way.

One advantage of the implementation we present is that it is robust against the destruction of any small random subset of the units or connections, but the real advantage (which we have not demonstrated in this simulation) comes from the ability of a connectionist network to do a rapid best-fit match. This is potentially much more powerful than the standard implementations which find all exact matches and then do conflict resolution. In situations where no existing rule fits perfectly, it may be sensible to apply a plausible rule, particularly in a learning system that needs to explore the space of plausible actions in order to find a satisfactory one. The ability of a connectionist implementation to settle on plausible but imperfect matches could therefore be very helpful, but only if the matching apparatus is able to do more than simple, variable-free "template" matches. Our eventual aim is to exploit the best-fit ability of DCPS to allow it to do more of the computation in each match so that it can perform complex tasks with fewer rule firings, and rules in one domain can be created by analogy with rules in other domains. But before we can do this we must demonstrate that it is possible to build a workable system that uses distributed representations and enforces consistent variable bindings during a match. So our current model only has a few of the interesting emergent properties that we eventually hope to demonstrate: It is damage-resistant, and the capacity of its working memory is dependent on the similarity of the items being stored.

2. A RESTRICTED CLASS OF PRODUCTION SYSTEMS

In order to simplify the task of implementing a production system in a connectionist network, we have made a number of restrictive assumptions about the production system. The working memory elements of DCPS are triples of symbols, such as **(F A B)**. We have chosen an alphabet size of 25 symbols, giving 25^3, or 15,625 possible triples. Only a few of these are present in working memory at any one time; typically there will be half a dozen. The sparseness of working memory is an important consideration in the design of the model.

Production rules in DCPS consist of two left-hand side clauses that specify triples and any number of right-hand side actions that modify working memory by adding or deleting triples. In the first version of DCPS no variables were allowed. In the second version, every rule contains two instances of a single variable. The variable must appear as the first element of both clauses on the left-hand side, and it can also appear anywhere on the right-hand side. So a typical rule is:

Rule-0: (=x A B) (=x C D) => +(G =x P) −(=x R =x)

In addition to these restrictions on the contents of working memory and the form of the production rules, we assume that during each match only one rule with one variable binding will be correct; this obviates the need for a conflict resolution mechanism. Naturally, we believe that all these restrictions can eventually be relaxed. Recent work by Touretzky and Krulwich (unpublished) on heterogeneous variable binding allows the variable to appear in either the first, second, or third position of each left-hand side clause, instead of always in the first position. This modification adds considerable flexibility but also permits an interesting kind of binding confusion error; details will be reported at a later date.

3. THE STRUCTURE OF WORKING MEMORY

The most straightforward representation for a set of triples would be a purely "localist" one, where every triple was represented by a dedicated unit. A unit in the active state would then indicate that the corresponding triple was present. We have rejected this idea in favor of a distributed or "coarse coded" representation (Hinton, 1981b; Hinton et al., 1986). Localist representations require too many units and too many connections; they quickly succumb to combinatorial explosion as the alphabet size or the length of a sequence increases. This is because localist representations do not make efficient use of the units when the number of items that are simultaneously present in working memory is much less than the number of possible items. Distributed representations use the information-bearing capacity of the units more efficiently by making them active much more often.[1] For an alternative view of distributed-versus-localist representations, see Feldman (1986).

[1] If there are 15,625 possible items, but only 6 of these are present at any one time, the probability that a working memory unit is active in a localist scheme is only about 0.0004. The average information conveyed by the unit is therefore the entropy of the distribution {0.0004, 0.9996} which is about 0.005 bits. In DCPS, fewer units are used to encode the same information, and each unit is active much more often so it conveys much more information. The probability of an individual unit being active is about 0.08 and so the average information it conveys is about 0.4 bits. However, in DCPS the correlation between units cannot be ignored (as it can in the previous case) and so the average information conveyed per unit is actually only about 0.04 bits.

In addition to the inefficiency of localist representations we think that a one-to-one mapping between individual neurons and symbolic structures is physiologically implausible; it is reminiscent of the grandmother cell idea. Recordings in the temporal lobe of the macaque cortex support the idea that neurons are tuned to very complex entities such as a face (Rolls, 1984) but they do not support the idea that a particular face is encoded by just one or just a few neurons. Each particular face is almost certainly encoded as a pattern of activity distributed over quite a large number of units, each of which responds to a subset of the possible faces. Using a distributed representation not only makes our model more efficient and neurally plausible, it also makes it tolerant of noise and occasional malfunctions.

3.1. Receptive Fields

The working memory space of DCPS, shown in Figure 1, consists of 2,000 binary state units. Each unit has a receptive field table such as the one in Figure 2. A unit's receptive field is defined to be the cross-product of the six symbols in each of the three columns, giving 6^3, or 216 triples per field. The unit described in Figure 2 has the triples (**C K R**) and (**F A B**) in its receptive field, along with 214 others. Receptive field tables are generated randomly prior to beginning the simulation; they determine the connection pattern between units in the various spaces comprising DCPS. Once the connections have been built and the working memory units' states have been initialized, the tables are no longer needed; they are not consulted when running the model.

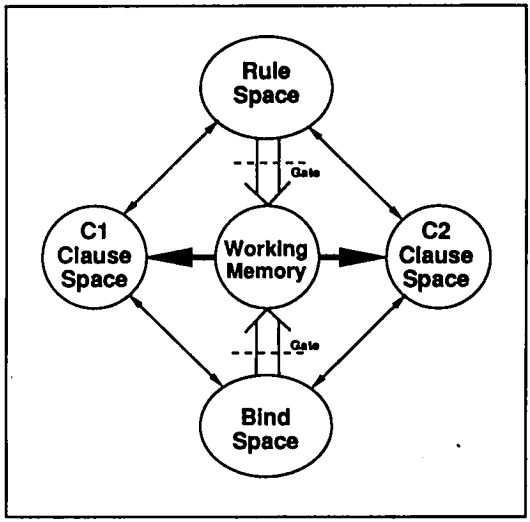

Figure 1. Block diagram of DCPS, a Distributed Connectionist Production System.

C	A	B
F	E	D
M	H	J
Q	K	M
S	T	P
W	Y	R

Figure 2. An example of a randomly generated receptive field table for a working memory unit. The receptive field of the unit is defined as the cross-product of the symbols in the three columns.

A triple may be stored in working memory by turning on all its receptors. With 2,000 working memory units, triples will average $6^3/25^3 \times 2{,}000 = 28$ receptors. The number of receptors per triple varies somewhat due to the random distribution of receptive fields. An external observer can test whether a particular triple is present in working memory by checking the percentage of active receptors for it. If this is close to 100%, the triple may be assumed to be present. For example, if the triple **(F A B)** were stored in working memory, the unit described in Figure 2 would be active, along with about 27 others. Although **(C K R)** also falls within the receptive field of this unit, the number of receptors two unrelated triples have in common is small; on average, it is less than one. Thus, while 100% of the **(F A B)** units become active when **(F A B)** is stored, only 1 out of roughly 28 **(C K R)** units would become active. To the external observer, **(F A B)** clearly is present in working memory and **(C K R)** clearly is not. But the network itself doesn't need to compute these percentages. It relies on the fact that triples that are present have strong effects and triples that are absent do not.

Figure 3 shows the state of working memory when the two triples **(F A B)** and **(F C D)** have been stored. The 2,000 working memory units are arranged in a 40×50 array, with the 55 that are active indicated by black squares. The positions of these 55 units in the array are not significant, since units' receptive fields are generated randomly. However, if we were to examine the receptive fields of each of the active units we would see that every one contains either **(F A B)** or **(F C D)**, or both.

Table 1 shows the first dozen triples with the strongest representations when working memory is in the state shown in Figure 3. **(F A B)** and **(F C D)** each have 100% of their receptors active, while the next best represented triple, **(F N B)**, has only 42% active. The average activity level over all 15,625 triples is much lower: only 2.7%. If we adopt the criterion that 75% of a triple's receptors must be active for it to be deemed present in memory, the division between present and absent triples in Table 1 is quite clear.

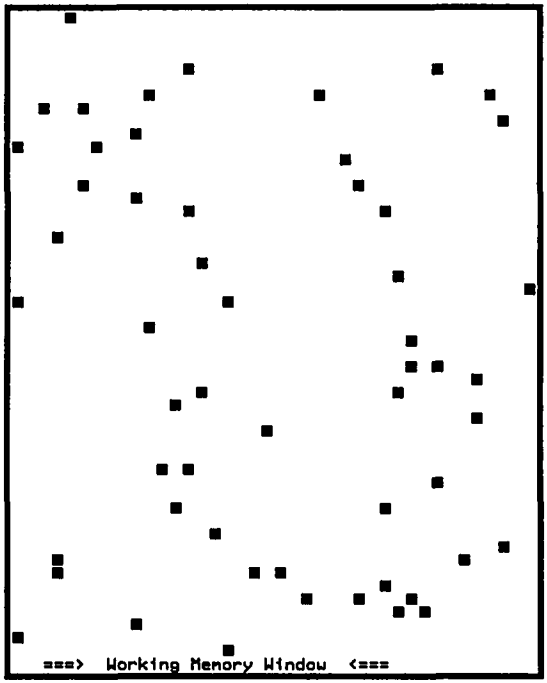

Figure 3. The state of working memory after the triples (**F A B**) and (**F C D**) have been stored. Active working memory units are indicated by black squares. 55 of the 2,000 units are active.

TABLE 1
The First Dozen Triples With the Strongest Representations
When Working Memory is in the State Shown in Figure 3.

	Levels of Triple Activation		
Triple	Percent Active (%)	Active Receptors	Total Receptors
(F A B)	100	28	/ 28
(F C D)	100	28	/ 28
(F A D)	40	11	/ 27
(F B D)	38	10	/ 26
(F A X)	37	11	/ 29
(S A B)	37	10	/ 27
(F Q D)	37	10	/ 27
(F C N)	37	10	/ 27
(F C B)	37	10	/ 27
(F C M)	35	10	/ 28
(F T D)	35	10	/ 28
(N C D)	34	10	/ 29

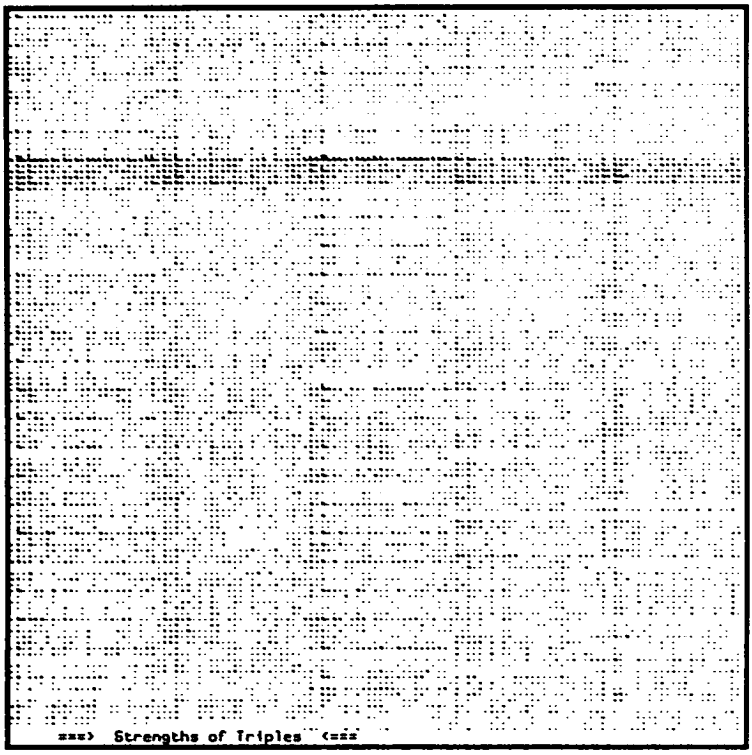

Figure 4. The levels of support for all 15,625 possible triples when working memory contains (F A B) and (F C D), represented by the 55 active receptors in Figure 3. The size of each blob indicates the number of active receptors for that triple.

Figure 4 shows the levels of support for all 15,625 possible triples when working memory contains (F A B) and (F C D). In the figure, (A A A) is located in the upper left corner and (Y Y Y) in the lower right. The blobs in this figure are associated with triples, not units; the size of each blob indicates how many receptors are active for that triple. A simple thresholding operation yields Figure 5, in which the (F A B) and (F C D) blobs stand out clearly and there is only a small amount of noise remaining.

3.2. Properties of Coarse Coding

Coarse coding representations have some useful properties and some psychologically interesting limitations. One useful property is tolerance of noise. If after storing some triples in working memory a few units are flipped on or off at random, the perceived contents of working memory will not be affected at all.[2] Tolerance of noise is especially important when items will be

[2] Assuming, of course, that we do not require strictly 100% of a triple's receptors to be active for it to be considered present.

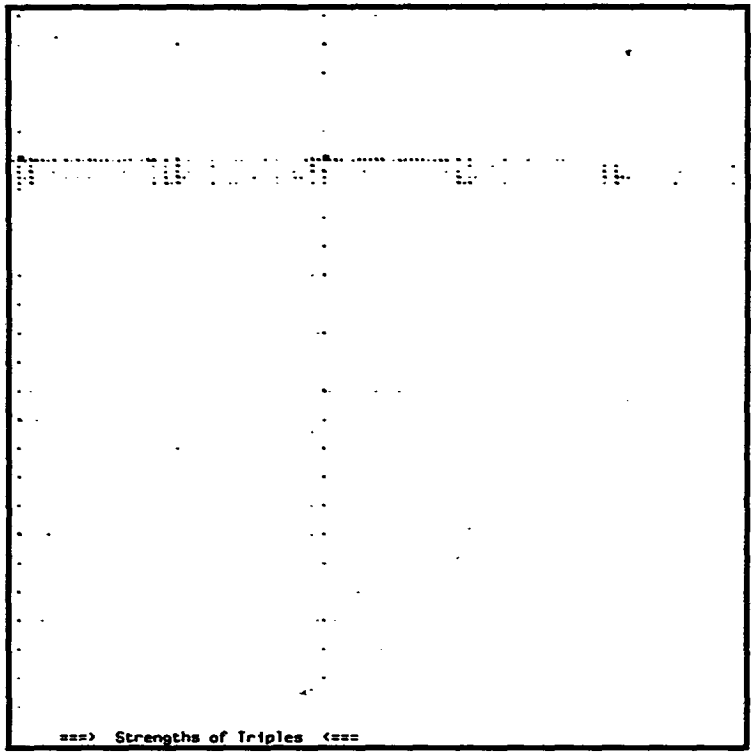

Figure 5. A moderately thresholded version of Figure 4. The (F A B) and (F C D) blobs stand out clearly here.

deleted from the memory as well as added to it. A slight overlap in the receptor set of related triples causes deletion of a triple to affect any related ones previously stored. That is, if (F A B) and (F C D) were stored in memory and (F C D) were then deleted by turning off all its receptors, it is likely that only 27 of the 28 (F A B) receptors would remain active, the 28th having been shared between the two.

The contents of working memory remain reasonably persistent because the overlap between any two triples is small. A visual effect resulting from this overlap can be seen in Figure 4. The dot pattern may appear completely random at first, but closer examination will reveal a regular series of thin horizontal and vertical bands. These bands are formed by triples that have two out of three components in common with the stored triples (F A B) and (F C D); on average such triples have seven of their receptors active, while triples with no components in common, such as (G S Q), average about 0.4 receptors active. Another effect that can be seen in the figure is the thick horizontal F band that is somewhat darker than the other bands. Since both of the stored triples begin with F, all other triples beginning with F have a slightly higher number of active receptors.

An interesting property of the coarse coded representation is that the memory has no fixed capacity; instead its ability to distinguish stored items from other items decreases smoothly as the number of stored items increases. Each triple added to working memory raises the number of active units, thereby increasing the support for other triples that have not been stored. As working memory fills up, the fraction of active receptors for certain triples that are "close" to those that have been stored approaches 100%, and the dividing line between present and absent triples blurs. If many closely related triples are stored, such as **(F A A)**, **(F A B)**, **(F A C)**, **(F A D)**, and so forth, then the system may exhibit local blurring, where it can't tell whether **(F A P)** is present or not but it is certain that **(G S Q)** is absent. Figure 6 illuslustrates the local blurring that occurs when four closely related triples are stored.

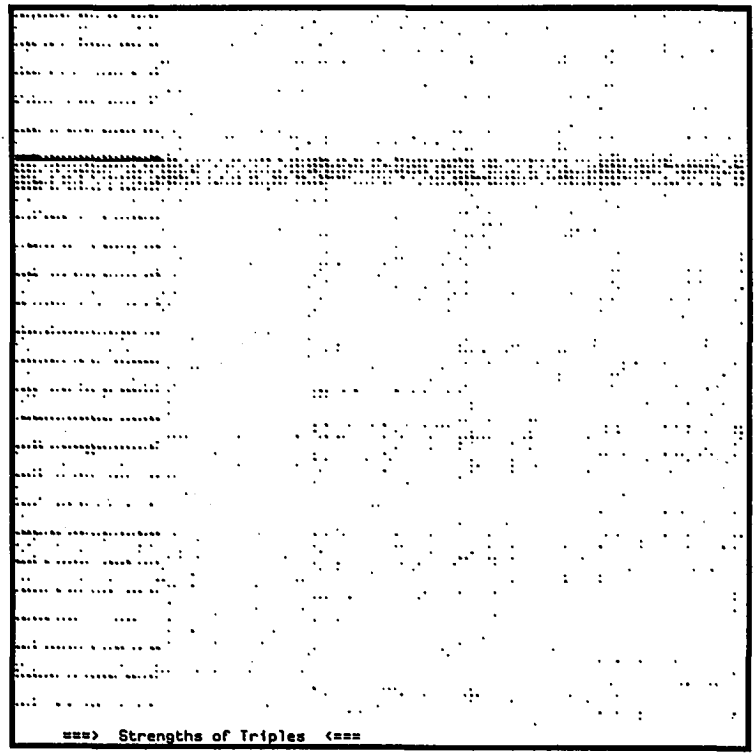

Figure 6. An illustration of the local blurring that occurs when several closely related triples are stored. Here, **(F A A)**, **(F A B)**, **(F A C)** and **(F A D)** have been stored. As a result, similar triples receive a high degree of support, as shown by the dark **(F A x)** line at the beginning of the F band and weaker **(x A y)** lines in the other bands. Moderate thresholding was applied.

Table 2 presents measurements of memory cross-talk under three different conditions as the number of stored triples increases. For each experiment we show the average percentage of support for the strongest triple not actually stored. In the first experiment we used maximally similar triples, such as (**F A x**) for various values of **x**. In the second experiment we used randomly generated triples, while in the last we used maximally dissimilar triples, such as (**A A A**), (**B B B**), (**C C C**), and so forth, which have no elements in common. It can be seen from this table that if triples are generated randomly, approximately 20 can be stored before the strongest non-stored triple achieves 75% support.

Triples stored early on in a coarse-coded memory will eventually fade away if rule firing deletes a large number of other triples. This gradual

TABLE 2
Average Percent Support for the Strongest Non-stored Triple after N Other Triples Have Been Stored in Memory.

	Crosstalk From Stored Triples		
	% Support for the strongest non-stored triple when the stored triples are:		
No. of Triples Stored	Maximally Similar	Randomly Generated	Maximally Dissimilar
1	39	39	38
2	55	41	41
3	69	45	43
4	74	48	45
5	86	49	47
6	91	50	48
7	95	53	50
8	98	56	52
9	99	58	54
10	100	60	55
11		61	56
12		64	57
13		66	59
14		67	59
15		69	60
16		69	62
17		71	63
18		73	63
19		73	65
20		74	66
21		75	66
22		76	68
23		76	70
24		77	70
25		78	70

decay phenomenon is again an effect of the overlap of receptive fields. Table 3 measures decay in the support of one stored triple as a series of other triples are deleted. (Even if a triple hasn't been stored, it can be deleted by turning off all its receptors. In this case, the only effect is to weaken the activation of triples with which it shares receptors.) One way to counteract the decay effect is to recall a triple before it has completely faded away, and then store it again. Whenever a triple is stored all its receptors become active, so its representation in working memory is refreshed.

4. SELECTIVE ATTENTION: CLAUSE SPACES

Clause spaces, labeled C1 and C2 in Figure 1, are a device for focusing the network's attention on particular triples from the set stored in working

TABLE 3

Average percent support remaining for a single stored triple after *N* other triples have been deleted in memory. The decay effect is due to sharing of receptors with the deleted triples.

	Crosstalk from Deleted Triples		
	% Support remaining for a single stored triple when the triples being deleted are, with respect to it:		
No. of Triples	Maximally Similar	Randomly Generated	Maximally Dissimilar
0	100	100	100
1	82	100	99
2	64	98	98
3	49	97	97
4	37	96	96
5	28	95	96
6	22	94	95
7	17	93	94
8	13	92	93
9	10	90	92
10	7	89	91
11	5	88	90
12	3	87	89
13	2	86	88
14	1	85	87
15	0	84	86
16		83	86
17		82	85
18		81	84
19		80	83
20		79	82
21		78	81
22		78	80
23		76	80
24		76	79

memory. Clause spaces are like the "pullout networks" which were used by Mozer (1987) to allow a perceptual system to attend to specific objects in a scene. The matching problem in DCPS consists of selecting the two triples in working memory that together satisfy the left-hand side of some production rule. Each clause is responsible for pulling out one of these triples.

There is a one-one excitatory mapping between working memory units and units in C1 and C2 spaces, so that each working memory unit that is active tries to turn on its corresponding C1 and C2 units. What prevents the C1 and C2 spaces from exactly copying the activity pattern in working memory is the fact that clause units are mutually inhibitory within their space, that is, each of the 2,000 C1 units inhibits the other 1,999 units, and similarly for C2; working memory units do not inhibit each other (see Figure 7). The inhibition level in clause space is carefully adjusted so that only about 28 units per space can remain active simultaneously, that is, just enough to represent a single, coarse coded triple. Exactly which triple is selected depends on various outside influences imposed on the clause space by units in the rule and bind spaces. Briefly, a clause unit will be able to remain active despite inhibition from its siblings only if it receives support from rule and bind units that are also active.

The apparent requirement that a clause space have $(N^2\text{-}N)/2$ bidirectional inhibitory connections might seem a flaw in the design, since as the number of units grows the number of connections quickly becomes unreasonable. With 2,000 clause units there would have to be 1,999,000 connections per space. But these connections need not actually be built. The inhibition function can be accomplished more economically by $2N$ unidirectional connections: N excitatory connections from clause units to a special regulatory

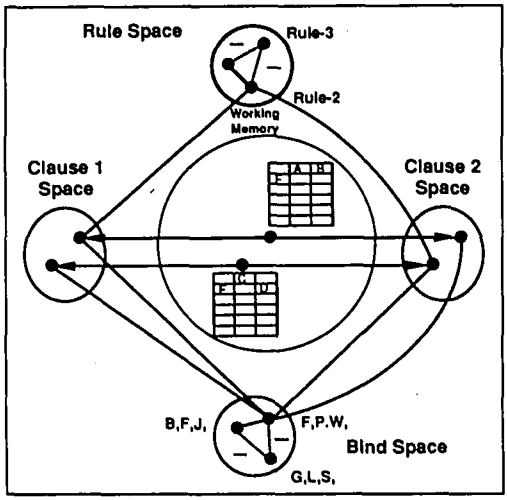

Figure 7. Connection pattern between clause units and working memory, rule, and bind units.

unit with a graded or integer-valued rather than binary response,[3] plus N inhibitory connections in the opposite direction. To mimic exactly the effect of $N(N-1)$ pairwise connections we would also need one excitatory connection from each unit to itself to cancel out the inhibitory effect it has on itself via the regulatory unit, giving $3N$ total connections. However, in practice these recurrent connections may be omitted with negligible effect. The fault tolerance of the inhibitory mechanism in the fully connected version of the network can be achieved by using several regulatory units, wired in parallel with proportionately smaller weights, rather than one unit with large weights.

For analysis purposes we will treat DCPS as an instance of a Hopfield network, and later, a Boltzmann machine. In order to meet those definitions we will ignore the regulatory unit solution and adopt the pretense, for the remainder of this article, that $(N^2\text{-}N)/2$ bidirectional inhibitory connections are actually built where required.

Note that although clause spaces are constrained to have roughly 28 units active at a time, not all patterns of 28 active units correspond to a valid triple. Clause spaces can sometimes be in an intermediate state where there are, say, 15 receptors for **(F A B)** active, 10 for **(G K Q)**, and 5 for something else. In other words, the clause-space units can divide their attention among several partly represented triples simultaneously. At higher temperatures (more relaxed constraints), more than 28 units can be active, which increases the chance that multiple triples will be partly represented. There is nothing analogous to this in conventional computers, where symbol structures remain discrete and must be considered one at a time (Derthick & Plaut, 1986).

5. THE RULES

5.1. Rule Format

Production rules in DCPS consist of two left hand side clauses that specify triples and any number of right hand side actions that modify working memory by adding or deleting triples. We first consider rules without variables. A typical rule would be:

Rule-1: **(F A B) (F C D) => +(G A B) +(P D Q) −(F C D)**

This rule can fire if **(F A B)** and **(F C D)** are both present in working memory. If it does fire, the triples **(G A B)** and **(P D Q)** will be added to memory and **(F C D)** will be deleted.

5.2. Representation of Rules

Each rule is represented by a population of 40 Rule units; the pattern of connections between these units and the clause units is determined by the

[3] These regulatory units resemble inhibitory interneurons which may play a similar role in cortex.

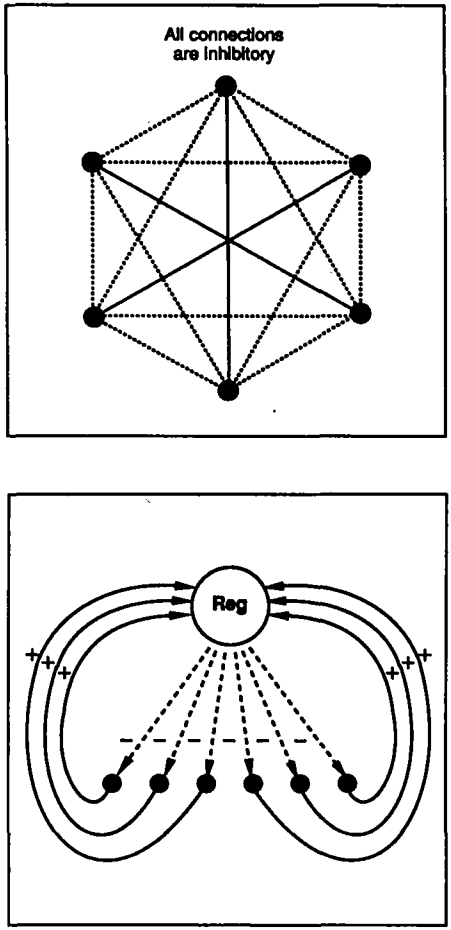

Figure 8. (a) Space of binary state units whose activity is limited by mutual inhibition using $(N^2\text{-}N)/2$ bidirectional connections. (b) Introduction of a regulatory unit with graded response accomplishes the same effect with only $2N$ one-way connections.

left-hand side of the rule. For example, Rule units that represent Rule-1 above will have bidirectional excitatory connections to C1 units whose receptive field includes (**F A B**), and C2 units whose receptive field includes (**F C D**), as shown in Figure 7. If a sufficiently large number of these C1 and C2 units become active, indicating that the triples (**F A B**) and (**F C D**) are present in working memory, the rule unit will also become active. Conversely, since the connections are bidirectional, when a Rule-1 unit becomes active it provides support for units in C1 and C2 space that support that rule.

The 40 units representing one production rule are connected so as to form a clique. Each active unit provides a slight excitatory stimulus to the

other units in its clique and a slight inhibitory stimulus to units in all the other cliques. Thus, Rule space is organized as a "winner-take-all" network (Feldman & Ballard, 1982); when the network settles, all the units in one clique will be active and all the remaining units will be inactive. This is how the system decides which rule to fire.

There are several reasons for implementing rules as collections of units rather than as individual units. First, it is damage resistant. Second, it allows binary units to give a graded response.[4] If, during the settling phase, there is a weak match between one rule and working memory, this will be indicated by only some of the corresponding rule units being active. If another rule matches more strongly, more of the units in its clique will be active, and they will eventually overpower the units in the other cliques. The implementation of rules in DCPS is "semidistributed": Rules are represented by the collective activity of a set of units, but each unit codes for only one rule. The problem with using a fully distributed rule representation is explained in section 11.3.

A further reason for implementing rules with multiple units is that it frees any one unit from having to represent the entire pattern associated with a rule's left-hand side. Each rule unit is connected to a random subset of all the clause units associated with the rule's left-hand side; only the clique as a whole has a complete representation for the rule. This is a more plausible organization than one in which rules are represented by single units, since it allows us to limit the connectivity of rule units without limiting the complexity of rules.

As in the case of clause spaces, the problem of building $O(N^2)$ connections among rule units can be solved by the use of regulatory units with graded outputs and a combination of one-way and bidirectional connections, as shown in Figure 9. Each rule unit excites its clique's "pro" regulatory unit, which in turn excites all its siblings in the clique; the unit also receives inhibition from its clique's "con" regulatory unit. The regulatory units of the various cliques are in turn connected to a master regulatory unit that controls the entire rule space. Each clique's pro unit has an inhibitory connection to the corresponding con unit to counterbalance the tendency for a clique to inhibit itself via the master regulatory unit. As in Figure 8, the recurrent connections from rule units to themselves, which are needed for absolute equivalence to the original network, have been omitted.

---

[4] One could implement rules as individual units with continuous rather than binary outputs, but the resulting network would not be a Hopfield net or Boltzmann machine. The fact that our hypothesized regulatory units have graded (either continuous or integer-valued) activation levels can be ignored because those units are merely used to simulate an equivalent Hopfield net composed solely of binary state units, with $O(N^2)$ rather than $O(N)$ connections.

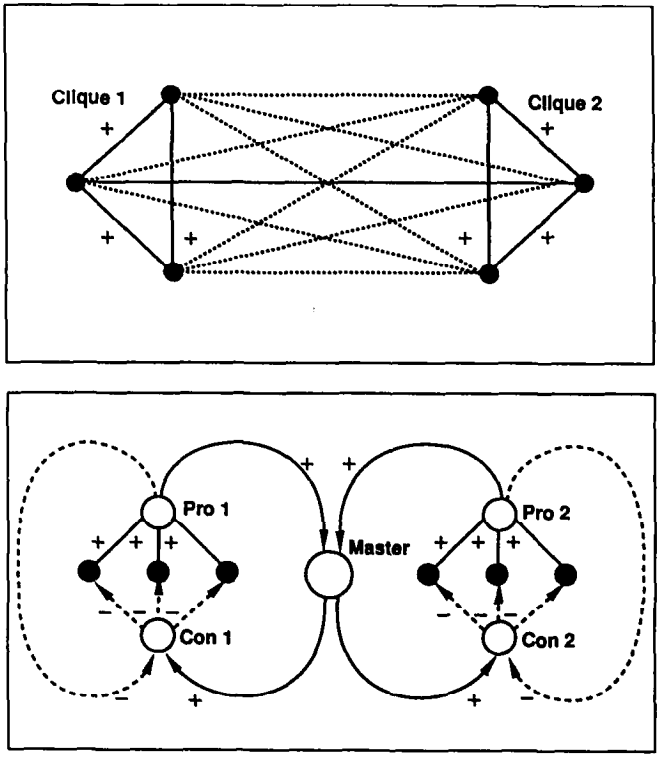

Figure 9. (a) A winner-take-all network composed of two cliques with three units each requires $(N^2\text{-}N)/2$ connections. (b) Use of regulatory units with graded response produces the same effect with only $2N+3C$ connections, where C is the number of cliques.

6. VARIABLE BINDING

6.1. Constraints on Rules

The first version of DCPS, called DCPS1, did not allow rules to contain variables. In developing DCPS2, which allows a limited form of variable binding, there were three distinct binding problems to consider:

1. Left-hand sides in which variables impose intraclause constraints, for example, the clause (=x R =x) can only match triples such as (F R F) or (G R G).
2. Left-hand sides in which variables impose interclause constraints. The pair of clauses (=x A B) and (=x C D) can match pairs of triples such as (F A B) and (F C D) or (G A B) and (G C D), but not (F A B) and (G C D).

3. Right-hand side actions in which variables appear. Variable binding requires a memory so that the variable's value can be instantiated into right hand side actions when the rule fires.

Each of these problems requires a different type of wiring pattern. Intraclause constraints appeared to be the least interesting, since they affect connections from rule space to just one of the two clause spaces. Therefore they were not included. DCPS2 does include a limited form of interclause constraint: each rule must have a variable in the first position of *both* left-hand side clauses.[5] DCPS2 also permits unrestricted use of variables on the right-hand side. A typical DCPS2 rule is:

Rule-2: (=x **A B**) (=x **C D**) => +(**G** =x **P**) −(=x **R** =x)

If this rule fires by matching (**F A B**) and (**F C D**), so that =x is bound to F, its right-hand side will add (**G F P**) to working memory and delete (**F R F**).

6.2. The Structure of Bind Space

Variable binding, which refers both to the imposition of constraints on rule matching and the instantiation of bound variables, is handled by the fifth space of units in Figure 1, the bind space.[6] The units in this space form a winner-take-all network with 25 cliques, one for each of the 25 symbols of the alphabet. The space is coarse coded, so that each unit belongs to three cliques (votes for three distinct symbols) rather than one. Since bind space contains a total of 333 units, each symbol falls in the receptive field of $(3/25) \times 333$ or 40 bind units, except for Y which has only 39.

Each bind unit has a set of bidirectional excitatory connections to units in C1 and C2 space whose receptive field table contains one or more of the letters the bind unit votes for. An F/J/W bind unit, for example, connects to a randomly chosen set of 240 C1 units: 80 that are receptors for triples beginning with F, 80 for J, and 80 for W. The same bind unit would also connect to a similar but independently chosen set of 240 C2 units. If a C1 unit that is a receptor for (**F A B**) and is connected to this bind unit becomes active, it will excite the bind unit, which in turn will excite other C1 and C2 units that code for triples beginning with F, J, or W. With many units in the F bind clique active, C2 space is more likely to adopt an activity pattern representing a triple beginning with F. The global effect of bind space is that it forces the C1 and C2 spaces to select triples beginning with the same symbol; that is how the "variable binding constraint" is imposed.

---

[5] This choice was arbitrary; we could have chosen to require that the variable appear, say, in the first position of clause 1 and the third position of clause 2. The important constraint is that the variable be in the same position in all rules.

[6] These bind units are similar to the mapping units used by Hinton (1981a) for representing viewpoint during object recognition.

The inhibitory connections between cliques in bind space prevent the number of active bind units from growing much above 40, which is just enough to activate all the units that vote for a particular symbol as the value of the bound variable. The stable states of this network (considered in isolation) each consist of one active clique of 40 units, with the remaining units inactive. But because each unit is a member of three cliques, in a stable state the winning symbol receives 40 votes whereas the 24 remaining symbols receive 3 to 4 votes each.[7] Even when bind space has settled on a value for the variable, it is still giving some slight consideration to other values. This consequence of the coarse coded representation may help the network avoid getting trapped in local minima when searching for a globally optimal rule match, though this issue needs further research.

6.3. Representation of Rules with Variables

Rule units do not participate in variable binding or the imposition of variable binding constraints on the match; they simply match a broader subset of working memory patterns than when the rule does not contain a variable. Thus the introduction of left-hand side variables "defocuses" the rule in some sense; the bind space imposes its own set of constraints on the model, orthogonal to those of rule space, that sharpens the match again.

In variable-free rules the two left-hand side clauses each specify roughly 28 units per clause space that must be examined to determine whether the rule should match. When a clause contains one variable instance, the clause as a whole can match any one of 25 possible triples. But the number of clause units that must be attended to in order to match any one of these 25 triples is far less than 25×28, because similar triples tend to share receptors. On average, the number of receptors for a clause with one variable instance, such as (=x A B), is $6^2/25^2 \times 2{,}000 = 115$.

It is not necessary to connect each rule unit to every relevant clause unit. In DCPS2, rule units connect to only 40 clause units out of 115, or about one-third of the relevant population. Thus when a rule is totally satisfied, each rule should be receiving activation from $28 \times (40/115) = 9.7$ units per clause space. The actual number will vary somewhat due to variance in the number of receptors per triple, variance in the distribution of active clause units across the different 40-unit subsets of clause space, and noise in the match. Since each rule unit connects to a different random subset of the relevant clause units, the collective activity of the entire clique of rule units tends to smooth over the noise and variance problems that individual units encounter.

---

[7] Each symbol is voted for by 40 units, and each unit votes for 3 symbols, so in a stable state there are 120 votes to be had. Since 40 go to the winner, the losers average $(120-40)/(25-1) = 3.3$ votes apiece.

Although DCPS2 does not include rules with intraclause variable binding constraints, such as (=x R =x), this has been considered as a possible extension. A clause that contains two instances of the same variable, like a clause that contains one, can match any of 25 possible triples. However, the number of receptors that must be considered is larger, because the 25 triples have only one element in common instead of two. An estimate of the number of receptors that match (=x R =x) is:

$$(6/25) \times [1 - \frac{19 \cdot 18 \cdot 17 \cdot 16 \cdot 15 \cdot 14}{25 \cdot 24 \cdot 23 \cdot 22 \cdot 21 \cdot 20}] \times 2{,}000 = 406$$

Experimentally we have found the value to be about 410. Since clauses with two instances of the same variable match an average of 410 clause units, versus 115 for clauses with one variable instance, rule units for two-variable clauses must look at a much larger subset of each clause space. This reduces each unit's capacity for discrimination and increases the variance in the activation it can receive during a match. These problems might be countered by invoking the law of large numbers, that is, raising the number of rule and clause units in the model to compensate for increased variance in the inputs of individual units.

7. THE MATCH PROCESS

So far we have described a network consisting of five spaces of units: working memory, C1, C2, rule, and bind. Working memory units are essentially latches; they do not perform computation, but their activity pattern drives the rule match process. C1, C2, rule, and bind units are wired up in complex but principled ways. Ignoring the possible use of regulatory units, all units have binary states, and all connections between units are bidirectionally symmetrical. The important questions to ask at this point are:

1. What are the stable states of such networks?
2. Under what conditions will a network eventually settle into one of its stable states?
3. Do stable states bear any relation to valid rule matches?

The first two questions have already been answered by Hopfield (1982). After reviewing these answers, we will try to present a convincing argument to deal with the third.

7.1. Hopfield Networks

A Hopfield network is a neural network composed of binary threshold units, all of whose connections are symmetrical. Hopfield proved that if units change state asynchronously and there are no transmission delays across connections, the network's stable states are those states α that minimize a

certain energy measure $E(\alpha)$. Let w_{ij} denote the weight of the connection between the ith and jth units; let θ_i denote the threshold of the ith unit; and let s_i^α denote the state (0 or 1) of the ith unit when the network as a whole is in state α. Then the energy of a state is the sum of the active units' thresholds minus the sum of the weights of connections between pairs of active units:

$$E(\alpha) = \sum_i s_i^\alpha \theta_i - \sum_{i<j} s_i^\alpha s_j^\alpha w_{ij}$$

This energy measure derives from an analogy Hopfield draws with spin glasses in physics, which operate under the same sorts of constraints as the neural networks he was studying. The stable states of these networks are called *local energy minima* because energy cannot be lowered any further by an individual unit's flipping state. Hopfield showed that networks that meet his constraints will settle into an energy minimum from any starting state because each state change either leaves the energy unchanged or reduces it; thus the energy decreases monotonically as the network moves from its initial state to a stable state. In general, however, the particular minimum energy state the network will end up in cannot be predicted from the starting state, and there is no guarantee that it will be a global minimum[8] rather than a local one.

7.2. Matching as Parallel Constraint Satisfaction

The argument that a valid rule match corresponds to a minimum energy state, in fact, to a global energy minimum, is based on reformulating the match as a constraint satisfaction problem. Weighted connections between units cause them to impose constraints on each other and the energy of a state is a measure of how much it violates the constraints. So a minimum energy state is one in which as many constraints are satisfied as possible. The following sorts of constraints are present:

- Due to their high thresholds, clause units cannot become active unless their corresponding working memory units are active.
- Due to mutual inhibition, only about 28 clause units can be active simultaneously in each space, which is just enough to represent one triple.
- Rule and bind units influence the clause units. A triple can remain active in C1 or C2 space only if it is supported by a population of rule and bind units; that is, it must match some rule's left-hand side and contain the symbol voted for by the active bind clique in its first position.
- Active clause units excite the rule and the bind units with which they are compatible. For example, C1 units whose receptive field includes **(F A B)** will try to turn on any rule units whose first clause is **(=x A B)**, and any bind units that support the variable **F**.

[8] A global minimum is a state whose energy is less than or equal to the energy of all other states the network could be in.

- Rule space is organized as a winner-take-all network. Rule units excite others that vote for the same rule and inhibit those that vote for different rules.
- Bind units form a coarse-coded winner-take-all network. They excite other units that vote for the same symbol (or symbols, if they have more than one in common), and inhibit units that vote for different symbols.

Considered individually, the C1, C2, rule, and bind spaces have many equivalent stable states. For instance, if bind space weren't connected to the clause spaces (which in turn are influenced by working memory), its 25 stable states would be completely equivalent; the network would then be equally likely to settle into any one of them. Rule space has as many stable states as there are rules; if rule space weren't connected to the clause spaces its various stable states would also be equivalent. But considered together, the spaces interact with each other, which means their stable states are not equivalent. The parameters have been set so that the only way all the model's constraints can be satisfied—thus putting the network into a global energy minimum—is for the C1 and C2 spaces to settle into representations of triples that are in fact present in working memory and that match some rule; for rule space to settle into a state where that particular rule is the winner; and for bind space to settle into a state representing the symbol in the first position of the triple in C1 space, which is also in the first position of the triple in C2 space. The derivation of the model parameters that govern the interactions between these spaces is discussed in section 9.

Constraint satisfaction in a Hopfield net is not a foolproof match technique because it is possible for the network to get stuck in a local energy minimum that does not represent a valid match. This occurs when a winner-take-all space, either rule or bind, settles so deeply into an undesirable stable state (e.g., all the units of one incorrect clique on, the remaining units off) that the other spaces cannot dislodge it.

In practice, the Hopfield net version of DCPS had no trouble finding the global energy minimum when the answer to the match problem was clear. However, in more difficult cases where there were many elements in working memory, many similar rules, or many partial matches possible but only one correct one, the network would often get stuck in a local minimum. In order to improve the chances of settling into the global minimum, DCPS was converted to a Boltzmann machine.

7.3. Boltzmann Machines

A Boltzmann machine (Ackley, Hinton, & Sejnowski, 1985) is a Hopfield network whose units behave stochastically as a function of their *energy gap*. A unit's energy gap is the amount by which its activation exceeds its thresh-

old. The energy gap of the ith unit when the rest of the network is in global state α, written $\Delta E_i(\alpha)$, is defined as:

$$\Delta E_i(\alpha) = (\sum_j s_i^{\alpha} w_{ij}) - \theta_i$$

While the deterministic units of a Hopfield network turn on whenever their energy gap is positive, that is, whenever their input exceeds their threshold, in a Boltzmann machine a unit's energy gap determines only the probability that it will turn on, in accordance with the Boltzmann distribution. Let $p_i(\alpha)$ denote the probability that the ith unit is in state 1 when the remainder of the network is in state α. This probability is given by the formula

$$p_i(\alpha) = \frac{1}{1 + e^{-\Delta E_i(\alpha)/T}}$$

The parameter T in the above equation is called the *temperature.* At very high temperatures units behave almost randomly; that is, the probability that a unit will turn on is approximately 0.5. (It is slightly above 0.5 for units with large positive energy gaps, slightly below 0.5 for units with large negative energy gaps.) On the other hand, when the temperature is close to zero the behavior of the units is almost deterministic; that is, the Boltzmann machine acts like a Hopfield net (see Figure 10). At moderate temperatures units tend to turn on when their energy gaps are positive, but they have a small probability of turning on even if their energy gap is negative, and a small probability of turning off even if their energy gap is positive. So at moderate temperatures a Boltzmann machine will occasionally move uphill in energy space, although the trend is still to move downhill. The higher the temperature the more likely an uphill move will be made.

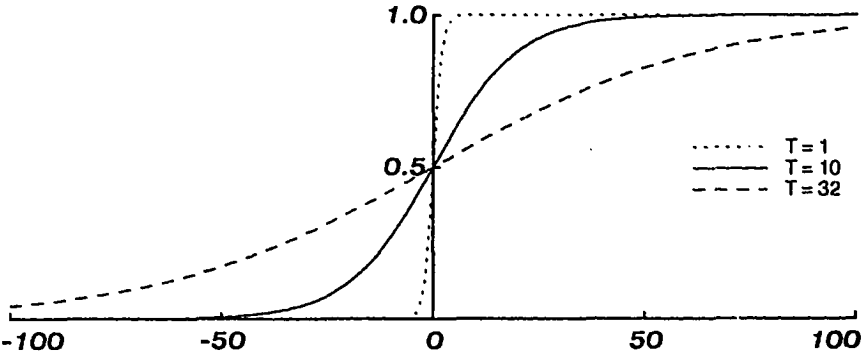

Figure 10. Graph of the Boltzmann equation for three different temperature values. This sigmoid curve shows the probability p_i that a unit will become active as a function of its energy gap ΔE_i.

If a Boltzmann machine starts out at high temperature and is very gradually cooled to a temperature close to zero, it is likely to end up in a state that is a global energy minimum. The probability that this will happen can be brought arbitrarily close to 1.0 by lowering the temperature sufficiently slowly (Geman & Geman, 1984). This stochastic search technique, which is known as simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), has been applied with good results to optimization problems unconnected with neural networks, and has also been applied to a variety of problems in low-level vision (Marroquin, 1985).

7.4. Matching by Simulated Annealing

The ability to move uphill in energy space allows the Boltzmann version of DCPS to escape local energy minima as it searches for the global minimum. In practice, we have not had to use a genuine annealing search in order to get acceptable performance from the network. When we ran the network at zero temperature, it got trapped in poor local minima, but we discovered that this could be avoided by running at three distinct temperatures. Figure 11 shows the temperature schedule used in the current version of the model.

The network is initialized for matching by turning off all rule, bind, and clause units, leaving it in a zero energy state. Next its state is "randomized" by running it at a relatively high temperature of 300 for two cycles. A cycle consists of 2,000 random updates of C1, C2, rule, and bind spaces. Although

1. **Initialize:** turn off all rule, bind, and clause units.
2. **Randomize:** run for 2 cycles at temperature 300. This temperature is high enough to ensure that all units which have any chance of being part of the solution have a reasonable chance of turning on, but it is low enough that completely irrelevant units are unlikely to be on.
3. **Match:** run for up to 10 cycles at temperature 33; stop if the energy is less than 2500 after any cycle.
4. **Cleanup:** run for 4 cycles at a temperature which is effectively zero. (We actually used 0.1 to avoid dividing by zero.)
5. **Reblas:** raise the thresholds of all clause, rule, and bind units by 50, 50, and 30, respectively.
6. **Verify:** run for 5 cycles at temperature of effectively zero.

Figure 11. The temperature schedule used in the Boltzmann machine version of DCPS.

the updating of units is done randomly, on average each of the 2,000 C1 and C2 units will get one chance to update its state; the rule and bind spaces, being smaller, are updated more frequently. Bind units average $2{,}000/333 = 6$ chances per cycle to update their state, and rule units average $2{,}000/40R$ chances, where R is the number of rules. In a six-rule system, rule units would average $2{,}000/240 = 8.3$ updates per cycle. The unequal update frequencies are an artifact of the way the update algorithm was programmed. While the particular ratios are not critical, it is probably useful to update the winner-take-all networks somewhat more frequently than the clause spaces, to ensure adequate rule and bind space response to each shift in the clause space patterns.

As Figure 10 shows, units behave fairly randomly at a temperature of 300, but they are still more likely to be active if their energy gap is positive than negative. At this temperature we have observed that the units that support the correct match and units that support partial matches are the ones that are on most often; units unrelated to a legal match become active less frequently. With so many units on, the energy of the network becomes quite high; with six rules (240 rule units) it varies between 12,000 and 17,000. See Figure 12.

The real matching work is performed in the next step of the schedule, at a temperature of 33. The precipitous drop in temperature from 300 to 33 is more suggestive of quenching than annealing but has no adverse effect on the match. The continued activity of rule, bind, and clause units now depends more strongly on support received from other units, but the network retains enough flexibility at this moderate temperature to explore various match possibilities rather than sink into the nearest local minimum. Cliques for a particular rule in rule space or symbol in bind space may become very active, fade away, and become active again. Triples may materialize in the clause spaces, be partly replaced by other triples, and then perhaps return. The

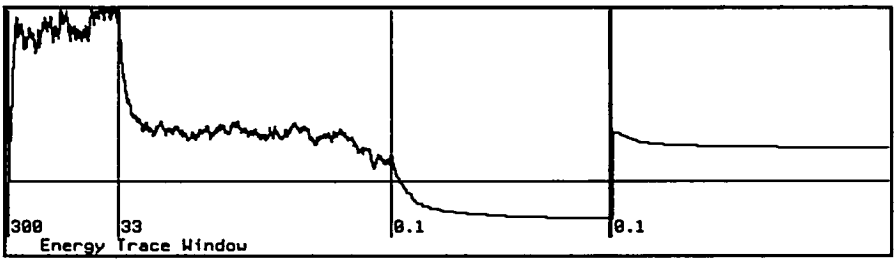

Figure 12. A graph of the energy level as the network follows the temperature schedule of Figure 11. Thin vertical divisions mark temperature changes, with the new temperature shown at the bottom of the graph. A thick division marks the point where thresholds are raised in the rebiasing step.

energy of the network rises and falls, but the general trend is decreasing. Once the energy falls below 2,500 or so,[9] the system is deep enough into a local minimum that it is unlikely to get out, so we move on to the cleanup step of the temperature schedule. In this step the network is run at a very low temperature, 0.1. Only units with positive energy gaps will remain active at this temperature. The result is that the clause spaces are left with roughly 28 units on, rule and bind spaces each have one clique active (40 units on), and the network is indicating as clearly as possible what it thinks the correct match should be.

7.5. Detecting Failed Matches

There are two ways in which the match can fail. The simplest is when the network fails to settle into any energy minimum at all. In this case very few of the units will have positive energy gaps, so when the temperature drops to 0.1 they will eventually all turn off. The more difficult case to detect is when the network has settled into a local energy minimum representing a partial match. The energy of a partial match is moderately negative. When the temperature drops to zero the network settles to the very bottom of the local energy minimum and stays there.

All correct matches have energies well below zero; this distinguishes them from partial matches, at least from the point of view of an external observer. Inside the connectionist model it is better if the behavior of individual units does not depend on measuring global properties such as energy. To detect failed matches without measuring energy directly, we raise the thresholds of the clause, rule, and bind units by 50, 50, and 30, respectively, in the rebiasing step of the temperature schedule. This is equivalent to applying an inhibitory bias of -50, -50, and -30, respectively, to each unit.[10] If a unit is getting sufficient support from its neighbors, its energy gap will remain positive and the unit will remain on. But in a partial match, at least one winner-take-all space is getting less than full support. For example, suppose Rule-2, whose left-hand side consists of (=x **A B**) and (=x **C D**), is matched against the triples (**F A B**) and (**G C D**). The Rule-2 clique can receive full support by pulling (**F A B**) into C1 and (**G C D**) into C2, but in this case neither the F nor G bind cliques can get full support. Suppose the F clique wins in bind space. It will receive full support only from C1 space. It gets slight support from C2 space because a few of the active (**G C D**) receptors will also be (**F C D**) receptors. There may also be a few (**F A B**) units turned on in C2 space since they are receiving support from working memory and the F bind clique, even though they are not supported by any connections

[9] This value is approximate and was determined empirically.

[10] These values were estimated empirically and are probably not optimal.

from rule space to C2 space. When the temperature goes to 0.1, the network settles to the bottom of this local energy minimum.

Now rebiasing is performed. Raising thresholds makes the energy gaps of units in the partly supported bind and C2 spaces negative, causing them to turn off. This in turn robs the rule and C1 units of support; as their energy gaps go negative, they turn off as well. The network ends up in the zero energy state in which all units are off. The partial match has been rejected. This outcome is illustrated by the energy trace in Figure 13.

One might wonder why the thresholds of the rule, bind, and clause units were not originally set at the higher level, eliminating the need for rebiasing. The reason can be seen in Figure 14, which shows schematically the effect of

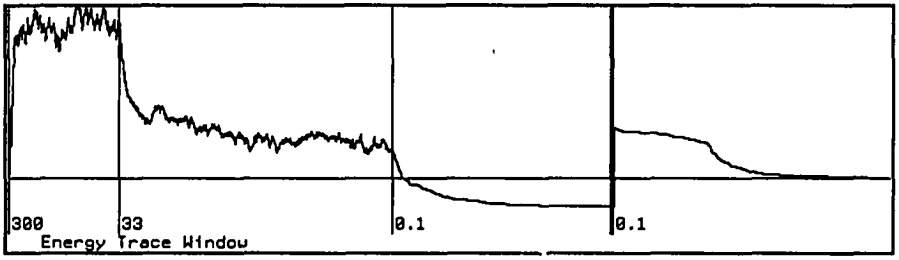

Figure 13. Detection of a partial match by rebiasing. Energy drops to zero as unit turns off after their thresholds are raised.

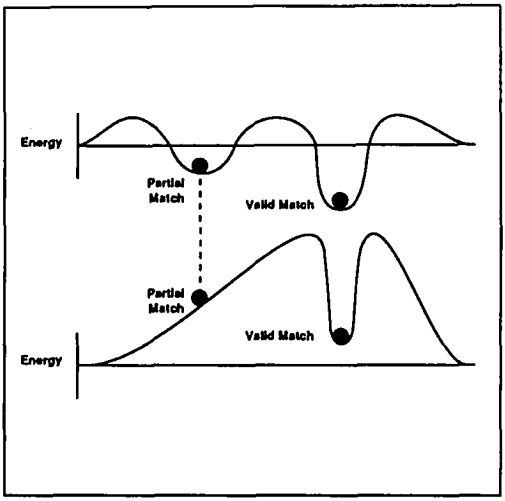

Figure 14. Effect of rebiasing on the energy landscape. The global energy minimum becomes a deep but narrower energy minimum. States representing local energy minima end up on a slope leading down to a zero energy state.

rebiasing on the energy landscape.[11] After rebiasing, the partial match state that was a local energy minimum is now located on a slope that leads down to the zero energy state with all units turned off. But the correct match state, formerly the global energy minimum, is now a local minimum whose energy is substantially positive. Thus, if the match were performed with the higher thresholds in place at the start, the network could find a better state simply by turning all its units off. When rebiasing is delayed until a low temperature has been reached, the network remains trapped in the state (now with positive energy, but still a local minimum) it was in if it managed to find the correct match.

Another effect of rebiasing visible in Figure 14 is that the energy well corresponding to a valid match is steeper and narrower than before, making it more difficult for the stochastic search to find if the network were not already in the well when rebiasing occurred. When the thresholds are raised with the model in a valid match state, the energy gaps of active clause, rule and bind units, although still positive, are left only slightly above threshold. The units are therefore critically dependent on support from all their neighbors. If the network moves any appreciable distance from this optimal point in state space it will no longer be in the energy well. Once the energy gaps of the active units go negative, the model will be more likely to move downhill to the zero energy state than uphill where it might fall back into the valid match energy well.

We have also considered the possibility of more flexible temperature schedules for coping with failed matches. After running for 10 cycles in the match phase at a temperature of 33, if the energy is not low enough for the network to have settled into the global minimum, it is probably in a state indicating a partly valid match. Either rule space has settled onto the right rule but bind space picked the wrong symbol, or else the reverse has occurred. To recover, we could run for a few cycles at a slightly higher temperature, around 40, to kick the network out of its local minimum, and then enter the match phase again.

8. RULE FIRING

After a rule has matched successfully it must be fired, which means performing its right-hand side actions that update working memory. The ability of rules to update a persistent symbol structure whose contents determine the next rule that will match is what enables DCPS to exhibit interesting sequential behavior. We first consider the problem of right-hand side update

[11] The true energy landscape is not continuous, nor can it be represented by a two-dimensional graph. It is an assignment of real values to the 2^N corners of an N-dimensional boolean hypercube representing the states of the network, where N is the number of units.

actions that are variable-free, and then move on to the general case where variables may appear in any position of a triple.

8.1. Variable-Free Actions

The right-hand sides of rules are implemented in DCPS as globally gated connections from rule units back to working memory units. The gate is closed during the match process so that rule units cannot affect working memory at all. During the rule firing portion of the production system's recognize–act cycle the gate is opened briefly; at this time, rule units that excite or inhibit working memory units can cause them to change their state. In the absence of outside stimuli, working memory units have a built-in hysteresis property that causes them to retain their current state. When the gate is closed prior to the next match cycle, working memory will be frozen in its updated state.

Consider the rule units that implement Rule-1. This rule adds the triples **(G A B)** and **(P D Q)** to working memory and deletes **(F C D)**. The units that implement Rule-1 will have gated excitatory connections to **(G A B)** and **(P D Q)** receptors, and gated inhibitory connections to **(F C D)** receptors. The hysteresis levels of working memory units are set so that no one rule unit can force them to change state; instead, the concerted action of several units is required. This is another feature of the model that contributes to its tolerance for unreliability in individual components: If a few random rule units fire spontaneously, they will have no effect on working memory.

Architectures with one-way and/or gated connections admittedly violate the definitions of a Hopfield network or a Boltzmann machine. DCPS requires these types of connections in order to produce sequential behavior; without them it would simply settle into an energy minimum and stay there. Fortunately these special connections only come into play during the rule firing phase of the recognize–act cycle. In the rule-matching phase the network is equivalent to one that is a pure Hopfield net/Boltzmann machine, because all the functioning connections are bidirectional and there are no gates opening or closing. The previously cited theoretical results of Hopfield and of Hinton and Sejnowski are therefore applicable to DCPS.[12]

8.2. Actions Requiring Instantiated Variable Values

To instantiate variable values into right-hand-side actions requires a cooperative effort between rule and bind spaces. Consider the **+(G =x P)** action in Rule-2. The Rule-2 units would collectively make excitatory connections to all working memory units that are receptors of **(G =x P)** for any value of **=x**. On average there will be $6^2/25^2 \times 2{,}000$, or roughly 115 such working

[12] A similar "equivalent network" argument can be made for the use of regulatory units, even though those units exert their influence during the match phase.

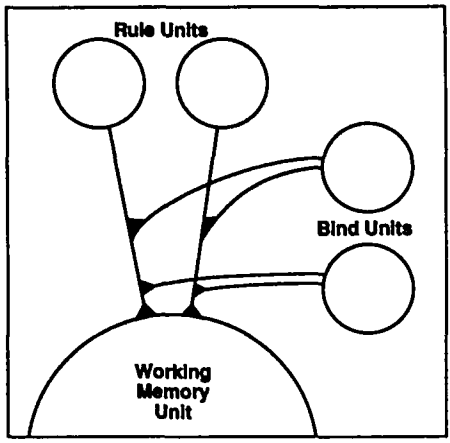

Figure 15. Right-hand-side actions involving variables are gated by excitatory connections from bind units onto the synapses that rule units make with working memory units.

memory units. However, these connections are individually gated by bind unit cliques: Connections (synapses) from rule units to working memory units are only effective if the connection itself receives some excitatory stimulation from bind units (see Figure 15). This is equivalent to saying that the input to working memory units is the conjunction of the activity coming from a rule unit and a clique of bind units.[13] Thus, if the network has settled into a state where the F clique is the winner in bind space, only connections from rule units to units that are receptors for **(G F P)** will be enabled. Each such connection from rule space back to working memory must be stimulated by several bind units in order to be effective; this is necessary because individual bind units vote for three different symbols; only the collection as a whole votes for a unique symbol. The requirement for support from multiple bind units also makes the network resistant to noise that could occur during rule firing due to randomly malfunctioning bind units.

Gated connections are also needed to allow actions to delete items from memory, because bind units by themselves have no way to tell whether the value they represent is needed for an add action or a delete action. In the case of delete actions such as **−(=x R =x)**, the connections from rule units

[13] The use of gated (or conjunctive) connections may appear to violate the normal ground rules of connectionist modeling. It is not difficult, however, to find biological structures that exhibit the crucial property of gated connections: a local nonlinear interaction between two synapses. Poggio and Torre (1978) have shown that such interactions can be expected to occur in the dendrites of cortical neurons, and Kandel and Schwartz (1982) have demonstrated the importance, in the sea slug *Aplysia,* of presynaptic facilitation, which is a different way of achieving local, nonlinear synaptic interactions.

to working memory are inhibitory, but the bind units' effects are always excitatory. By using gated connections, we allow the bind units to select the inhibitory connections that will be allowed to influence working memory.

8.3. Functions on Variable Values

Instead of instantiating the exact value of a variable into right-hand-side actions, we can instantiate some function of that value. The function will be "computed" by the gating pattern that bind units apply to the rule units' connections to working memory. For example, consider the increment and decrement functions. We will use >x and <x in right-hand-side actions to denote values one greater and one less than the value to which the variable is bound; for example, if the variable is bound to F, then <x appearing in a right-hand-side action would be instantiated as E and >x as G. Modular arithmetic should be used so that every symbol has a successor and predecessor: The successor of Y is A, and the predecessor of A is Y.

Figure 16 shows how the increment function could be used to step through a series of working memory elements sequentially by bumping a counter. The left-hand-side pattern (=x R R) refers to the counter value, which is maintained as a triple in working memory and incremented by a right-hand-side action. On successive firings, rule Seq-1 will step through the triples (A R R), (B R R), (C R R), and so forth, and leave behind another trail of triples (A B A), (B C B), and so on.

The implementation of the increment and decrement functions is straightforward. In actions that don't compute functions of the bound variable, such as −(=x R R), the rule units make connections to all working memory units that could match the action, and each connection is gated by bind units of the appropriate type. For example, a connection from a rule unit to an (A R R) unit would be gated by a set of bind units that vote for A, while a connection to a (B R R) unit would be gated by bind units that vote for B. To compute a function on the right-hand side, the connections that implement the right-hand-side action are simply gated by bind units specified by the function. Thus, in rule Seq-1, the increment function that appears in +(>x R R) can be implemented by using bind units that vote for A to gate (B R R) connections, bind units that vote for B to gate (C R R) connections,

Rule:

Seq-1: (=x R R) (=x R R) ⇒ −(=x R R) +(>x R R) +(=x >x =x)

Initial contents of working memory:

(A R R)

Figure 16. Use of right-hand-side increment function to step a counter.

and so on. Any mapping from symbols to symbols can be computed in this way.

9. DERIVATION OF MODEL PARAMETERS

In this section we review all the parameters of the model. We first distinguish arbitrarily chosen parameter values from theoretically well-motivated ones, and then demonstrate the combination of mathematical analysis and empirical testing that led to the derivations of the latter.

The 25-letter alphabet size and the 2,000-unit working memory size are obviously arbitrary. We chose triples for the working memory elements because they are relatively simple structures, yet complex enough to represent any directed graph. Semantic networks are often encoded as triples for this reason; for example, the triple (A R B) can encode a link of type R from A to B. Any structure simpler than triples would rob the model of representational power; structures larger than triples pose no theoretical difficulty for DCPS, but require exponentially greater computing resources for a given alphabet size.

The receptive field table size, six letters per column, represents a compromise between conflicting desires. On the one hand, having a lot of receptors for each triple makes the model robust by providing immunity to noise. The number of receptors per triple can be increased by increasing the size of receptive field tables (going to "coarser" units). On the other hand, moving to coarser units reduces the number of triples that can be stored simultaneously before cross-talk becomes excessive. The mathematics of this tradeoff are discussed by Rosenfeld and Touretzky (1988).

Another consideration in designing the working memory, which also occurs in several other places in the model, is the need to minimize variance from the expected value of a parameter. In the present scheme, where the expected number of receptors per triple is 28, when receptive fields are randomly generated the actual number of receptors per triple is binomially distributed with substantial variance. The variance in the number of receptors per triple must be small for the model to work reliably. If one triple has 19 receptors and another has 40, even when the first triple is 100% active it will appear weaker than the second triple at 60% activity. One way to reduce this variance would be to increase the number of units, but that was deemed impractical due to the memory limitations of the Lisp Machine. Instead we attacked the problem by generating pseudorandom receptive field tables that give the same mean number of receptors per triple as the random approach, but far less variance. The procedure we used to generate the table is described in Appendix B. A third way to reduce the variance would be to make each unit coarser, subject to the memory capacity tradeoff mentioned previously.

In bind space, the decision to have each bind unit vote for 3 letters was also a compromise. On the one hand, the larger the receptive field of a bind unit, the fewer the number of units needed; on the other hand, if the receptive fields are too large the model will lose its ability to discriminate one letter from another. Given our purely arbitrary decision that each letter should be supported by a clique of 40 units, a local winner-take-all implementation would require 1,000 bind units total. Letting each bind unit vote for 3 letters reduces the number of units to 333 (with only 39 units for Y instead of 40) without harming the model's accuracy. The collective effect of 40 coarse-coded bind units voting for one letter outweighs the $40 \times (3-1) = 80$ votes they generate for the other 24 letters, since the latter get an average of just 3.3 votes each. The decision that rule cliques should have 40 units each was also arbitrary.

The next set of choices involve the connectivities of individual units. Recall that the expected number of clause units that match a clause containing a single variable, such as (=x **A B**), is $(6/25)^2 \times 2{,}000 = 115$. We arbitrarily decided to allow each rule unit to connect to 40 units in C1 space and another 40 in C2 space, meaning it covers approximately one-third of the population specified by the rule's left-hand side.

To wire up the bind units, we note that each letter appears in column one of $(6/25) \times 2{,}000 = 480$ units' receptive field tables. We arbitrarily decided to allow each bind unit to connect to 80 units in C1 space and another 80 in C2 space for each of its three letters, yielding a total of 240 connections to each clause space. For simplicity in programming the model, the 80 connections were chosen independently for each letter, so a bind unit could potentially have as many as three connections to one clause unit if they had more than one letter in common. Two hundred and forty connections is again substantially less than the entire population of $3 \times 480 = 1{,}440$. The reduced connectivity demonstrates that the model as a whole can produce correct behavior even when individual units have only limited knowledge of the global state.

The most difficult parameters to derive were the weights for the various connection types. The match process demands a delicate balance among a number of constraint types, so that no one constraint will dominate or be ignored. There is only one inviolable constraint: A clause unit should never turn on if its corresponding working memory unit isn't on. This was assured by giving the clause units very high thresholds, and the connections from WM to the clause spaces very high weights. We chose a value of 900; all other weights in the model are integers less than 10. (The model is restricted to integer weights for speed.)

Before proceeding to the derivation of the remaining weights, we first consider what the network will look like when it has found a perfect match and is sitting in what should be a global energy minimum. There will be 40

rule units, 40 bind units, and on average 28 C1 units and 28 C2 units active. Even though the number of active rule and bind units is equal, the connections these two types of units make to the clause spaces do not deserve equal weight, as the combinatorial analysis below indicates.

In a perfect match, the average number of connections an active clause unit will have to active rule units is $(40/115) \times 40 = 13.9$. The average number of connections an active clause unit will have to the active bind units is:

$$(80/480) \times 40 \times [1 + 5/24 + 5/24] = 9.4$$

The 5/24 terms account for the fact that bind and clause units both vote for more than one letter. Consider the clause unit whose receptive field was given in Figure 2. The first column of its receptive field table is C/F/M/Q/S/W. If it is active because **(F A B)** is the winning triple in that clause space, most of its input from bind space will come from bind units that include the C/F/M/Q/S/W unit among their 80 F connections, such as an F/J/W bind unit. But the F/J/W unit might have another connection to the clause unit because they both vote for W. And other bind units, such as an F/M/V unit, that do not happen to include this clause unit among their 80 F connections, might still vote for it because they have another letter in common: in this case, M. These coincidental effects cannot be ignored; they account for roughly 29% of the clause unit's input from bind space.

To balance the influence of rule and bind units on clause units, we set the weight of connections between rule and clause space to $+5$, and the weight of connections between bind and clause space to $+7$. Then the average rule input of $13.9 \times +5 = +69.5$ closely matches the average bind input of $9.4 \times +7 = +65.8$.

Next there is the issue of lateral inhibitory and excitatory weights within the two winner-take-all spaces. Each rule unit excites its siblings, which vote for the same rule, and inhibits rival units which vote for other rules. The competitive effect must be strong enough to force the space into a stable state where only one clique is active, but not so strong as to overpower the influence of the clause spaces in selecting the proper clique to be the winner. This consideration led us to set the rule unit sibling excitation weight at $+2$, and the rival inhibition weight at -2. Returning to the case of the ideal match, a rule unit in the winning clique will recieve an excitation of $39 \times +2 = +78$ from its siblings. It will also be connected to roughly $40 \times (28/115) = 9.7$ active clause units in each clause space, but there is substantial variance in this figure since the rule units' connections to clause space are determined by picking random subsets of the 115 units. The expected excitation from the two clause spaces is $9.7 \times +5 \times 2 = +97$, giving the rule unit a net input of $+175$. By comparison, a rule unit with only one of its two left-hand-side clauses satisfied will receive excitation of $+49$, and inhibition of -80 from the winning rule clique, giving a net input of -31.

In order to set the thresholds of rule units we must again take variance into account. To make the model robust we chose a lower bound of five active connections per clause space rather than the expected 9.7. When rebiasing has been performed there will be an additional inhibitory input of -50, so after rebiasing the worst case net input to a rule unit in the winning clique is $(2\times 5\times +5)+(39\times +2)-50=+78$. We therefore set the rule unit's threshold at 69, giving it an energy gap of $+9$ in the worst case and $+56$ in the average case. Besides the need for a positive energy gap, another reason for setting the threshold somewhat below the worst-case activation level is that during the early part of the match phase there may be even fewer units on for the correct rule than at the end of the worst-case correct match; there is also likely to be some inhibitory input from rule units in other cliques that haven't died out yet.

The bind space weights and thresholds were determined by an analogous procedure. First, the expected number of connections between an active bind unit and the active units in one clause space is:

$$(80/480)\times 28\times[1+5/24+5/24]=6.6$$

Due to variance, we set the lower bound on a bind unit's activation from clause units at 4.5 from each space, or 9 units total. We set the weights between bind units in the same clique (i.e., units that have at least one letter in common) to $+1$. This gives bind units in the winning clique a net input of $(2\times 4.5\times +7)+(39\times +1)=+102$. Rebiasing contributes an (empirically chosen) value of -30. Therefore we set the bind unit's threshold at $+63$, giving an energy gap of $+9$ in the worst case and $+68$ in the average case. A value of -2 for the inhibitory weight between bind units in different cliques was determined by experimentation.

Although many of the model's weights and thresholds can be justified analytically in retrospect, in building the model we relied heavily on experimentation and measurement. Our initial combinatorial analyses were less sophisticated than what is presented here, and we made no attempt to predict variance, only means. In practice it suffices to use the mean inputs to a unit for an initial estimate of the correct weights, thresholds, and rebiasing parameters, and then tune these values empirically to account for variance and other factors.

Finally, we come to the issue of lateral inhibition in the two clause spaces. The inhibition must be set high enough to allow only about 28 units per clause space to remain active; the units that are selected will be the ones that receive the most support from the winning rule and bind cliques. We chose a lateral inhibition weight of -2 for clause space units. In an ideal match, each clause unit will receive an input of $+900$ from its corresponding working memory unit, $+5$ from each of approximately 13.9 rule units, $+7$ from each of approximately 9.4 bind units, -2 from each of approximately 27

other clause units, and −50 from the rebiasing, yielding a net input of 931. We therefore set the clause unit threshold at 901 to leave room for variance in the rule and bind inputs, and to maintain a positive energy gap.

10. EXPERIMENTAL RESULTS

10.1. Measured Performance

DCPS has run a 6-rule loop overnight through more than 1,000 rule firings without error. Working memory contained two triples at a time, and each rule firing involved one addition and one deletion. In the current version of the model, a rule match takes about 90 seconds on a Symbolics 3600 running Common Lisp. Part of this time is spent updating a graphic display as each unit changes state, so that the network's progress can be monitored during the match.

The number of rules the system can represent appears to be limited only by synergistic effects (explained shortly) and by the number of possible partial matches during each search. The largest production system we have run to date, which used a slightly modified version of DCPS as part of a parse tree manipulation task, had 17 rules (Touretzky, 1986b).

The matching portion of an annealing typically involves two to six probes of each clause unit, where a probe consists of computing the unit's energy gap, deciding whether or not it will change state, and notifying its neighbors if its state does change. Failed matches are detected after 10 cycles, when the cleanup portion of the temperature schedule is begun.

10.2. Difficult Match Cases

Early in the development of DCPS we adopted the simplifying assumption that match problems would always have unique answers, so that only one rule and one variable binding could constitute a valid match. This allowed us to avoid the issue of conflict resolution (Brownston, Farrell, Kant, & Martin, 1985), which, although interesting, is not central to our current enterprise. But even with this simplifying assumption some match problems are more demanding than others, and situations can be contrived in which DCPS has difficulty finding the correct solution. Two such situations are discussed below.

In the simplest match cases there are no partial matches to worry about; the triples in working memory that do not match the winning rule do not match any of the other rules either. In more complex cases several feasible-looking matches exist with relatively low energy states; the system is forced to search among them to find the lowest one. This involves calling up different triples in the clause spaces for each possibility. As the number of partial matches increases, DCPS becomes more likely to settle into a local minimum representing a partly successful match rather than finding the lowest energy

state associated with the one correct match. Figure 17 shows a set of rules and working memory elements that produce this behavior. In theory, annealing long enough and slowly enough would solve the problem, since the correct match is always a deeper energy minimum than any partial match.

In this match scenario there are six triples in working memory; the clause spaces must select from among the 36 possible ways to form a pair of triples the one combination the produces a correct match. What makes this problem difficult is the fact that four pairs of triples have fairly deep energy minima representing almost-successful partial matches (see Table 4). In these partial matches, either both clauses on the left-hand-side of Rule Comb-1 or Comb-2 are satisfied but the variable binding constraint is not, or else only one of the left-hand-side clauses is satisfied but the variable binding constraint is met because both clause spaces support the same bind clique (J or K.) The source of the combinatorial confusion is the fact that all three rules and all three bind cliques are capable of getting full support from the clause spaces, so it's difficult to choose among them; what differentiates partial from complete matches is the fact that rule and bind space can't *both* get full support except when the rule is Comb-3 and the variable =x is bound to M.

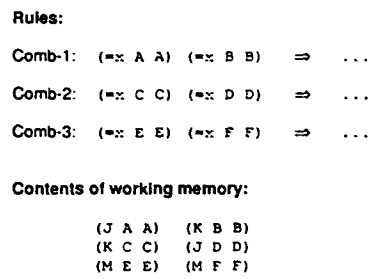

Figure 17. A match situation in which combinatorial complexity hinders the search for a valid match.

TABLE 4

The four partial matches generated by the rules in Figure 17 have fairly deep energy minima, but there is a global minimum, representing the one complete match, in which all constraints are satisfied.

		Partial and Complete Matches		
Degree of Match	Triple in Clause 1	Triple in Clause 2	Rule Supported	Binding Supported
Partial	(J A A)	(K B B)	Comb-1	half J, half K
Partial	(J A A)	(J D D)	half Comb-1, half Comb-2	J
Partial	(K C C)	(J D D)	Comb-2	half J, half K
Partial	(K C C)	(K B B)	half Comb-1, half Comb-2	K
Complete	(M E E)	(M F F)	Comb-3	M

DCPS does not search a combinatorial space by sequentially enumerating the possibilities. The partial representations of competing triples coexist simultaneously in the clause spaces, while rule and bind winner-take-all spaces host similar competitions. The stochastic nature of the Boltzmann machine causes some competitors in a space to fade out, and possibly fade back in again, until the network as a whole settles deeply enough into an energy minimum that a clear winner emerges in each space.

Figure 18 illustrates another contrived case where it is difficult for DCPS to conclude the match correctly. **(M J J)** is present in working memory but none of the rules Syn-1 through Syn-4 can match, due to their second clause. While all rules compete with each other as a result of being in a winner-take-all network, the Syn rules also *help* each other by supporting **(M J J)** as the first clause. This unwanted synergy, which occurs whenever failing rules have related left-hand sides, interferes with the search for the correct match. In order to find this match, the lone Anti rule must override the four Syn rules and get the pattern for **(M R R)** into C1 space. The more Syn rules there are to support **(M J J)**, the harder this will be.

11. DISCUSSION

11.1. Alternative Implementations of Working Memory

There are two broad approaches to implementing a working memory in a connectionist network. The obvious method, which we use here, is to set aside a separate group of units whose activity encodes the current contents of working memory. A less obvious alternative is to use temporary modifications of connection strengths to make it easier to recreate patterns of activity that have recently occurred. The advantage of this second method is that it does not require any extra units to act as a memory, and the memory is automatically content-addressable—recent patterns can be reconstructed

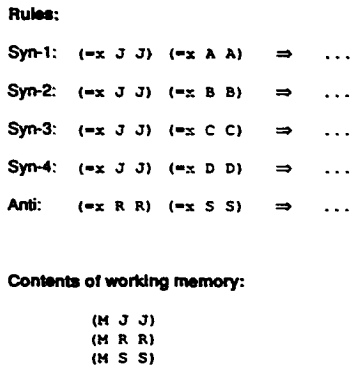

Figure 18. A match situation in which synergistic action between four rules that generate partial matches can prevent the system from finding the correct match.

from any sufficiently large subpattern. A particularly simple version of the second method is to implement working memory by temporarily lowering thresholds. In DCPS1, for example, the only effect of the units in the working memory space is to provide additional input to units in C1 and C2 space, so we could remove the working memory units and exactly mimic their effects by temporary reductions of the thresholds of units in the clause spaces. This would also get rid of all the one-to-one connections between the working memory and clause spaces.

One disadvantage of using thresholds instead of units is that each time a new item is inserted (or deleted) it is necessary to lower (or raise) thresholds in both clause spaces, because there is no way of knowing in advance whether the item will subsequently match the first or the second clause of a rule.

Some important properties of the working memory are broadly independent of whether it is implemented as activity levels, temporary threshold changes, or temporary weight changes. Because the working memory for each item is distributed over many units, thresholds, or weights, there will be interference if more than a few items are stored at once, and the interference will be greater as the items become more similar. This is a necessary consequence of using distributed representations to allow many more possible items than there are storage sites. We interpret the well-known limitations of human short-term memory as an indication that it too may involve the use of distributed representations.

11.2. Multiple Interacting Distributed Representations

In the introduction we alluded to a problem that arises when there are interactions between several groups of units that each use distributed representations. Each unit takes part in the representation of many different items and its causal effects on units in other spaces must reflect this fact. This means that a unit in one space will generally provide excitatory input to a great many units in another space, and so there is a danger that the activation within each space will become more and more diffuse as time progresses. In DCPS, the tendency for activation to become more diffuse is counteracted by using lateral inhibition within the spaces. This suppresses units that are only supported by a small fraction of the units in other spaces and concentrates the activation on units which receive multiple excitatory inputs.

Winner-take-all networks, bind spaces, clause spaces (or pullout networks), and coarse-coded symbol representations are generally useful bits of machinery that have been profitably incorporated into other connectionist models. Touretzky (1986a) describes a system for manipulating recursive data structures, called BoltzCONS, that was assembled by rearranging the components of DCPS. BoltzCONS has only one pullout network instead of two, but it has three independent bind spaces. The bind spaces disassemble a triple into its component symbols. Gated connections were later introduced between the bind spaces of BoltzCONS and DCPS to allow the two

networks to pass symbols back and forth in a rule-based tree manipulation task (Touretzky, 1986b).

11.3. Why Rules Are Semidistributed

Working memory, clause, and bind spaces all use fully distributed representations, but the representation of rules in DCPS is "semidistributed." Rules are represented by collections of units, but each unit is associated with a only a single rule rather than being coarse coded. Sharing units between similar rules is counterproductive in this architecture, because rules with similar left-hand sides may have totally dissimilar or even directly opposed right-hand sides. Consider the two rules Sim-1 and Sim-2 below: One tries to add the triple **(H H H)** and one tries to delete it. The rule units common to Sim-1 and Sim-2, which should be in the majority because the rules are so similar, would have both excitatory and inhibitory connections to **(H H H)** working memory units. Thus, the majority of the rule units would have no action at all. More sophisticated versions of DCPS, which we are presently considering, may be able to exploit similarity among rules by segregating left-hand-side and right-hand-side operations into different collections of units.

Sim-1: **(=x A B) (=x C D) => +(H H H)**
Sim-2: **(=x B B) (=x C D) => −(H H H)**

11.4. Similarity and Generalization

One automatic consequence of using distributed representations is that similar items tend to have similar effects. This is a helpful effect if the particular distributed patterns that are used impose a similarity metric that reflects the important distinctions in the domain. If, for example, "cheese" and "chalk" have rather different representations but "cheese" and "cheddar" have rather similar representations, a connectionist network will tend to make sensible generalizations (Hinton et al., 1986). There have been many demonstrations of this effect when the experimenter chooses the distributed representations (Hinton, 1981c; Rumelhart & McClelland, 1986). More recently, Hinton (1986) has described a network that can construct the appropriate distributed representations for itself, so the generalizations cannot be said to have been determined by the experimenter.

DCPS does not currently make any use of similarity between triples or between rules, and it therefore fails to make good use of the properties that a connectionist implementation could provide. We view DCPS as only the first step in the development of connectionist symbol manipulation architectures. Future advances should lead to models which make better use of the powerful constraint satisfaction and generalization abilities of connectionist networks. Such models would be more than mere implementations of conventional symbol processing ideas because the connectionist substrate

would provide important computational properties that are not available in standard implementations.

11.5. Seriality and Variable Binding

DCPS is implemented in a massively parallel network and yet is it is unable to bind the variables in more than one rule at a time. It can perform a parallel search over rules that contain variables to discover which rule fits the contents of working memory best and during this search it considers many different rules and many different variable bindings in parallel, but it is unable to represent particular *conjunctions* of rules and variable bindings. Its only method of representing such a conjunction is by settling on a *single* rule and a *single* binding of each variable. This means that it is using simultaneity to represent the binding, and simultaneity cannot be used for representing several different bindings at once.

Many different variable bindings could be explicitly represented at the same time if we dedicated a separate unit to each possible conjunction of a rule and a variable binding, but this is equivalent to eliminating variables altogether by having many different, variable-free versions of each rule. Newell (1980) has advanced the idea that variable binding may be one of the things that forces people to be sequential processors, and DCPS corroborates this view. By separating the rule space from the bind space we achieve great economies in the number of units required, but the cost is that the only way to represent explicitly which binding goes with which rule is to settle on one bound rule at a time.

APPENDIX A. MODEL PARAMETERS

DCPS is one of the largest connectionist models built to date. Tables A-1 through A-3 give the number of units in each space and the types, numbers, and weights of their connections. (The table gives total numbers of connections, not numbers of connections from active units during a valid match.) Thresholds are expressed as connections with weight $-\theta$ to a "true unit" whose state is always 1.

TABLE A-1
Parameters of Clause Units

Clause Spaces: 2000 units each

Source of Connections	Number of Connections	Weight per Connection	
Working memory unit	1	+900	
Other clause units	1999	−2	*(mutual inhibition)*
Rule units	avg. 7 per rule	+5	
Bind units	avg. 40	+7	
True unit	1	−901	*(threshold)*

TABLE A-2
Parameters of Rule Units

Rule Space: 40 units per rule

Source of Connections	Number of Connections	Weight per Connection	
C1 clause units	40	+5	
C2 clause units	40	+5	
Sibling rule units	39	+2	
Rival rule units	40 per rival rule	−2	
WM units (gated)	40 per RHS action	*n/a*	
True unit	1	−69	*(threshold)*

TABLE A-3
Parameters of Bind Units

Bind Space: 333 coarse coded units: 3 letters per unit, 40 units per letter.

Source of Connections	Number of Connections	Weight per Connection	
C1 clause units	240	+7	
C2 clause units	240	+7	
Sibling bind units	avg. 107	+1	
Rival bind units	avg. 225	−2	
True unit	1	−63	*(threshold)*

APPENDIX B. GENERATING RECEPTIVE FIELDS FOR WORKING MEMORY UNITS

In our simulation, each triple in working memory is represented by activity in about 28 units. We initially chose the receptive fields of working memory units at random in the obvious way: Six different random letters are chosen for the first position, six for the second, and six for the third. Unfortunately this introduces large sampling errors. Triples represented by as few as 20 or as many as 36 active units are quite common. This can make it hard to distinguish between triples that are present but have few units to represent them and triples that are absent but have accidental activation in some of their many units. If the expected number of active units per triple was much larger than 28, the law of large numbers would eliminate this problem, but in our simulation we used a heuristic method for making the number of units per triple be more uniform.

We started with a set of receptive fields that were chosen so that every letter occurred equally often in each of the three positions. We then considered all possible triples, and recorded how many units encoded each triple. We defined a cost function which was the sum (over all possible triples) of the

square of the difference between $6^3/25^3 \times 2{,}000 = 27.648$ and the number of units encoding the triple. This measure is minimized when the number of units per triple is as uniform as possible. We performed gradient descent in this cost function by selecting moves which reduced the cost function but preserved the number of times a letter occurred in each position. A candidate move consisted of taking the receptive fields of two units and swapping two letters in corresponding positions. If, for example, two letters from the second position are swapped, the two receptive fields

```
((A B C D E F)  (G H I J K L)  (M N O P Q R))
((T U V W X Y)  (P Q R S T U)  (A C E G I K))
```

might become

```
((A B C D E F)  (G H R J K L)  (M N O P Q R))
((T U V W X Y)  (P Q I S T U)  (A C E G I K))
```

Candidate moves were selected at random, and were accepted whenever they reduced the cost function or left it unaltered. This was continued until no more improvements were encountered. We considered using simulated annealing to improve the solution, but simple gradient descent was already rather slow and it gave an adequate solution. The standard deviation was reduced from 4.9 to 1.5.

■ Original Submission Date: January 1987;
Resubmission and Acceptance December 1987

REFERENCES

Ackley, D.H., Hinton, G.E., & Sejnowski, T.J. (1985). A learning algorithm for Boltzmann machines. *Cognitive Science, 9,* 147–169.

Ballard, D.H., & Hayes, P.J. (1984, June). Parallel logical inference. *Proceedings of the Sixth Annual Conference of the Cognitive Science Society* (pp. 114–123). Boulder, CO.

Ballard, D.H. (1986). Parallel logical inference and energy minimization. (Tech. Rep. No. TR 142). Computer Science Department, University of Rochester, Rochester, NY.

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

Brownston, L., Farrell, R., Kant, E., & Martin, N. (1985). *Programming Expert Systems in OPS5.* New York: Addison-Wesley.

Derthick, M.A., & Plaut, D.C. (1986, August). Is distributed connectionism compatible with the physical symbol system hypothesis? *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 639–644). Amherst, MA.

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

Feldman, J.A. (1986). Neural representation of conceptual knowledge. (Tech. Rep. No. TR-189). Department of Computer Science, University of Rochester, Rochester, NY.

Geman, S., & Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 6,* 721–741.

Hinton, G.E. (1981a, August). A parallel computation that assigns canonical object-based frames of reference. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* Vol. 2 (pp. 683–685). Vancouver, BC, Canada.

Hinton, G.E. (1981b, August). Shape representation in parallel systems. *Proceedings of the Seventh International Joint Conference on Artificial Intelligence,* Vol. 2 (pp. 1088–1096). Vancouver, BC, Canada.

Hinton, G.E. (1981c). Implementing semantic networks in parallel hardware. In G.E. Hinton & J.A. Anderson (Eds.), *Parallel Models of Associative Memory.* Hillsdale, NJ: Erlbaum.

Hinton, G.E. (1986, August). Learning distributed representations of concepts. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 1–12). Amherst, MA.

Hinton, G.E., McClelland, J.M., & Rumelhart, D.E. (1986). Distributed representations. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the microstructure of cognition,* volume 1. Cambridge, MA: Bradford Books/MIT Press.

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

Kandel, E., & Schwartz, J. (1982). Molecular biology of memory: Modulation of transmitter release. *Science, 218,* 433–443.

Kirkpatrick, S., Gelatt, C.D., & Vecchi, M.P. (1983). Optimization by simulated annealing. *Science, 220,* 671–680.

Marroquin, J.L. (1985). Probabilistic solution of inverse problems. (Tech. Rep. No. AI-TR-860). MIT Artificial Intelligence Laboratory, Cambridge, MA.

Mozer, M.C. (1987). *The perception of multiple objects: A parallel, distributed processing approach.* Doctoral dissertation, University of California, San Diego.

Newell, A. (1980). Harpy, production systems and human cognition. In R.A. Cole (Ed.), *Perception and production of fluent speech.* Hillsdale, NJ: Erlbaum.

Poggio, T., & Torre, V. (1978). A new approach to synaptic interactions. In R. Heim & G. Palm (Eds.), *Approaches to complex systems.* Berlin: Springer.

Rolls, E.T. (1984). Neurons in the cortex of the temporal lobe and in the amygdala of the monkey with responses selective to faces. *Human Neurobiology, 3,* 209–222.

Rosenfeld, R., & Touretzky, D.S. (1988). Scaling properties of coarse-coded symbol memories. In D.Z. Anderson (Ed.), *Neural information processing systems.* (Collected papers of the IEEE Conference on Neural Information Processing Systems—Natural and Synthetic. Denver, CO, November 1987.) New York: American Institute of Physics.

Rumelhart, D.E., & McClelland, J.L. (1986). *Parallel distributed processing: Explorations in the microstructure of Cognition.* Volume 1. Cambridge, MA: Bradford Books/MIT Press.

Touretzky, D.S. (1986a, August). BoltzCONS: Reconciling connectionism with the recursive nature of stacks and trees. *Proceedings of the Eighth Annual Conference of the Cognitive Science Society* (pp. 522–530). Amherst, MA.

Touretzky, D.S. (1986b, October). Representing and transforming recursive objects in a neural network, or "Trees *do* grow on Boltzmann machines." *Proceedings of the 1986 IEEE* International Conference on Systems, Man, and Cybernetics (pp. 12–16). Atlanta, GA.