Complex Systems **2** (1988) 463–484

Coarse-Coded Symbol Memories and Their Properties

Ronald Rosenfeld*
David S. Touretzky†‡
Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract.

Coarse-coded symbol memories have appeared in several neural network symbol processing models. They are static memories that use overlapping codes to store multiple items simultaneously. In order to determine how these models would scale, one must first have some understanding of the mathematics of coarse-coded representations. The general structure of coarse-coded symbol memories is defined, and their strengths and weaknesses are discussed. Memory schemes can be characterized by their *memory size*, *symbol-set size*, and *capacity*. We derive mathematical relationships between these parameters for various memory schemes, using both analysis and numerical methods. We find a simple linear relationship between the resources allocated to the system and the capacity they yield. The predicted capacity of one of the schemes is compared with actual measurements of the coarse-coded working memory of DCPS, Touretzky and Hinton's distributed connectionist production system. Finally we provide a heuristic algorithm for generating receptive fields which is efficient and produces good results in practice.

1. Introduction

A *distributed representation* is a memory scheme in which each entity is represented by a pattern of activity over many units [1]. If each unit participates in the representation of many entities, it is said to be *coarsely tuned*, and the memory itself is called a *coarse-coded memory*.

Coarse-coded memories have been used for storing symbols in several neural network symbol processing models, such as the Rumelhart and McClelland verb learning model [2], Touretzky and Hinton's distributed connectionist production system DCPS [3,4], Touretzky's distributed implementation of

*Computer addresses: `roni@cs.cmu.edu` (Internet).
†Computer addresses: `dst@cs.cmu.edu` (Internet).
‡Reprint requests should be sent to the second author, at the address above.

Lisp S-expressions on a Boltzmann machine, BoltzCONS [5,6], and St. John and McClelland's PDP model of case role defaults [7]. In all of these models, memory capacity was measured empirically (if it was measured at all), and parameters were adjusted by trial and error to obtain the desired behavior. We are now able to give a mathematical foundation to these experiments by analyzing the relationships among the fundamental memory parameters.

There are several paradigms for coarse-coded memories. In a *feature-based representation*, each unit stands for some semantic feature. Binary units can encode features with binary values, whereas more complicated units or groups of units are required to encode multi-valued properties or numerical values from a continuous scale. The units that form the representation of a concept define an intersection of features that constitutes that concept. Similarity between concepts composed of binary features can be measured by the Hamming distance between their representations. In a neural network implementation, relationships *between* concepts are implemented via connections among the units forming their representations. Certain types of generalization phenomena thereby emerge automatically.

A different paradigm is used when representing points in a multidimensional continuous space [8,1]. Each unit encodes values in some subset of the space. Typically the subsets are hypercubes or hyperspheres, but they may be more coarsely tuned along some dimensions than others [9]. The point to be represented is in the subspace formed by the intersection of all active units. As more units are turned on, the accuracy of the representation improves. The density and degree of overlap of the units' receptive fields determines the system's resolution [10].

Yet another paradigm for coarse-coded memories, and the one we will deal with exclusively, does not involve features. Each concept, or symbol, is represented by an arbitrary subset of the units, called its *pattern*. Unlike feature-based representations, individual units do not determine the meaning of a symbol. Only the pattern as a whole is assigned a meaning.

A symbol is stored in memory by turning on all the units in its pattern. A symbol is deemed present if all the units in its pattern are active.[1] The *receptive field* of each unit is defined as the set of all symbols in whose pattern it participates. We call such memories *coarse-coded symbol memories* (CCSMs). We use the term "symbol" instead of "concept" to emphasize that the internal structure of the entity to be represented is not involved in its representation. In CCSMs, a short Hamming distance between two symbols does not imply semantic similarity, and is in general an undesirable phenomenon.

The difference between CCSMs as defined above and dynamic memories of the type studied by Hopfield [11] should be emphasized. A Hopfield network stores patterns in the weights between active units. The units' outputs evolve

[1]This criterion can be generalized by introducing a *visibility threshold*: a fraction of the units in a pattern that should be on in order for a symbol to be considered present. Our analysis deals only with a visibility criterion of 100%, but can be generalized to accommodate noise.

over time as the network settles into an attractor state representing a single stored item. In contrast, CCSMs have no attractor dynamics; they are simply a coding scheme. External mechanisms such as the pullout networks of DCPS and BoltzCONS may be used for associative retrieval from a CCSM, but they are not part of the CCSM itself. The primary advantage of CCSMs is their ability to represent multiple items simultaneously, possibly using fewer resources than conventional, non-distributed schemes.

Coarse-coded symbol memories can be further classified by the degree to which they are structured. In a completely unstructured CCSM, any subset of the units is a legitimate candidate for representing a symbol. A structured CCSM, on the other hand, imposes restrictions on the class of patterns that may be used. These restrictions can be articulated in terms of the patterns themselves or in terms of constraints on the receptive fields of the units. Some constraints are very simple, e.g., that all patterns be of the same size. The working memory of Touretzky and Hinton's DCPS is a CCSM with more complex constraints. "Symbols" in this memory are triples of letters. The receptive field of each unit is generated by the cartesian product of three randomly-chosen sets of six letters each. Thus each 216-element receptive field is a $6 \cdot 6 \cdot 6$ subspace of a larger three-dimensional space, rather than a collection of 216 independently chosen symbols. Likewise, the Rumelhart and McClelland verb learning model stores triples of phonemes using cartesian product receptive fields. Rumelhart and McClelland refer to this as "conjunctive coding". In their model the fields are of varying size, rather than uniform as in DCPS.

Imposing structure (i.e., constraints) on receptive fields might be expected to reduce the capacity of the memory. When we measured this effect for DCPS by comparing its memory capacity to that of similar non-structured CCSMs, we found the actual penalty to be slight.

CCSMs can be very efficient for implementing large, sparse memories. By "large" we mean memories that are capable of representing many distinct symbols, and by "sparse" we mean that only a small fraction of these symbols will be simultaneously present in the memory. An extreme localist representation, in which each symbol is encoded by one unit and each unit is dedicated to encoding a single symbol, is very inefficient in such cases. For a given number of symbols, α, a localist representation requires exactly α units, whereas a CCSM can make do with far fewer than that. Alternatively, the advantage can be recast in terms of representational power: given N units, a localist representation can represent exactly N symbols, whereas a CCSM can potentially handle many more. The efficiency with which CCSMs handle sparse memories is the major reason they have been used in connectionist systems, and hence the major reason for studying them here.

The unit-sharing strategy that gives rise to efficient encoding in CCSMs is also the source of their major weakness. Symbols share units with other symbols. As more symbols are stored, more and more of the units are turned on. At some point, some symbol may be deemed present in memory because all of its units are turned on, even though it was not explicitly stored: a *ghost*

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
U_1	•			•	•		•	
U_2		•	•		•	•		
U_3		•		•	•			•
U_4	•					•	•	
U_5			•					•
U_6	•	•		•	•		•	

Figure 1: A memory scheme. 8 symbols are assigned overlapping patterns over 6 units. The columns are the symbols' *patterns*. The rows are the units' *receptive fields*.

is born. Ghosts are an unwanted phenomenon arising out of the overlap among the representations of the various symbols. The emergence of ghosts marks the limits of the system's *capacity*: the number of symbols it can store simultaneously and reliably.

In what follows, we define coarse-coded symbol memories rigorously, and develop a formalism in which questions about the performance of these systems can be given exact, quantitative formulation. Four different memory schemes are described, their capacities analyzed, and their strengths and weaknesses contrasted with one another. The principle of "economy of scale" is discussed, and the conditions under which it applies are spelled out. Structured symbol memories are presented next, where two examples from the literature (Touretzky and Hinton's DCPS and Rumelhart and McClelland's verb learning model) are analyzed. Actual capacity measurements of DCPS are compared with one of the theoretical schemes. Finally, we provide a heuristic algorithm for generating receptive fields which is efficient and produces good results in practice.

2. Definitions and fundamental parameters

A coarse coded symbol memory in its most general form consists of:

A set of **N** binary state **units.**

An alphabet of α **symbols** to be represented. Symbols in this context are atomic entities: they have no constituent structure.

A memory scheme, which is a function that maps each symbol to a subset of the units — its **pattern.** The **receptive field** of a unit is defined as the set of all symbols to whose pattern it belongs (see figure 1).

The exact nature of the memory scheme mapping determines the properties of the memory, and is the central target of our investigation.

As symbols are stored, the memory fills up and ghosts eventually appear. It is not possible to detect a ghost simply by inspecting the contents of memory, since there is no general way of distinguishing a symbol that was stored from one that emerged out of overlaps with other symbols. (It is sometimes possible, however, to conclude that there are no ghosts. This is true when every symbol that is visible in memory has at least one unit that is not shared with any other visible symbol.) Furthermore, a symbol that emerged as a ghost at one time may not be a ghost at a later time if it was subsequently stored into memory. Thus the definition of a ghost depends not only on the state of the memory but also on its history.

Some memory schemes guarantee that no ghost will emerge as long as the number of symbols stored does not exceed some specified limit. In other schemes, the emergence of ghosts is an ever-present possibility, but its probability can be kept arbitrarily low by adjusting other parameters. We analyze systems of both types. First, two more bits of notation need to be introduced:

P_{ghost}: Probability of a ghost. The probability that at least one ghost will appear after some number of symbols have been stored.

k: Capacity. The maximum number of symbols that can be stored simultaneously before the probability of a ghost exceeds a specified threshold. If the threshold is 0, we say that the capacity is **guaranteed.**

A localist representation, where every symbol is represented by a single unit and every unit is dedicated to the representation of a single symbol, can now be viewed as a special case of coarse-coded memory, where $k = N = \alpha$ and $P_{\text{ghost}} = 0$. Localist representations are well suited for memories that are not sparse. In these cases, coarse-coded memories are at a disadvantage. In designing coarse-coded symbol memories we are interested in cases where $k \ll N \ll \alpha$. The permissible probability for a ghost in these systems should be low enough so that its impact can be ignored, i.e., $P_{\text{ghost}} \ll 1$.

We wish to find memory schemes that will maximize the number of symbols α and the capacity k while minimizing N, the number of units required. We are also interested in the tradeoff between α and k for a fixed N. In the following section, we present four memory schemes, and analyze each of them in terms of the mathematical relationship among N, α, k, and P_{ghost}.

3. Analysis of four memory schemes

3.1 Bounded overlap (guaranteed capacity)

If we want to construct the memory scheme with the largest possible α (given N and k) while guaranteeing $P_{\text{ghost}} = 0$, the problem can be stated formally as:

> Given a set of size N, find the largest collection of subsets of it such that no union of k such subsets subsumes any other subset in the collection.

This is a well known problem in Coding Theory, in slight disguise. Unfortunately, no complete analytical solution is known. We therefore simplify our task and consider only systems in which all symbols are represented by the same number of units (i.e., all patterns are of the same size). In mathematical terms, we restrict ourselves to constant weight codes. The problem then becomes:

> Given a set of size N, find the largest collection of subsets of size *exactly* L such that no union of k such subsets subsumes any other subset in the collection.

We wish to provide two arguments in support of this simplification. First, we believe it does not significantly reduce the size of the collection. This is because the solution to the original problem is likely to be composed of subsets of similar size. This can be seen by considering the effect too small or too large a subset would have on the capacity of the system. An unusually small subset will have a very high tendency to become a ghost, whereas an unusually large subset will have a high tendency to create one.

The second argument is a pragmatic one. In order for coarse-coded memories to be useful, they need to be accessed by some external mechanism. One such mechanism is the clause space of DCPS. Clause spaces use lateral inhibition to extract a single stored symbol from a coarse-coded memory. This competitive mechanism works best when patterns are of uniform size.

There are no known complete analytical solutions for the size of the largest collection of patterns even when the patterns are of a fixed size. Nor is any efficient procedure for constructing such a collection known. We therefore simplify the problem further. We now restrict our consideration to patterns whose pairwise overlap is bounded by a given number. For a given pattern size L and desired capacity k, we require that no two patterns overlap in more than m units, where:

$$m = \left\lfloor \frac{L-1}{k} \right\rfloor . \tag{3.1}$$

Memory schemes that obey this constraint are guaranteed a capacity of at least k symbols, since any k symbols taken together can overlap at most $L-1$ units in the pattern of any other symbol — one unit short of making it a ghost. Based on this constraint, our mathematical problem now becomes:

> Given a set of size N, find the largest collection of subsets of size exactly L such that the intersection of any two such subsets is of size $\leq m$ (where m is given by equation (3.1).)

Coding theory has yet to produce a complete solution to this problem, but several methods of deriving upper bounds have been proposed (see for example [12]). The simple formula we use here is a variant of the Johnson Bound. Let α_{bo} denote the maximum number of symbols attainable in memory schemes that use bounded overlap. Then

$$\alpha_{bo}(N, L, m) \leq \frac{\binom{N}{m+1}}{\binom{L}{m+1}}. \tag{3.2}$$

The Johnson bound is known to be an *exact* solution asymptotically (that is, when $N, L, m \to \infty$ and their ratios remain finite).

Since we are free to choose the pattern size, we optimize our memory scheme by maximizing the above expression over all possible values of L. For the parameter subspace we are interested in here ($N < 1000$, $k < 50$) we use numerical approximation to obtain:

$$\alpha_{bo}(N, k) = \max_{L \in [1,N]} \frac{\binom{N}{m+1}}{\binom{L}{m+1}} < \max_{L \in [1,N]} \left(\frac{N}{L-m}\right)^{m+1} < e^{0.367\frac{N}{k}}. \tag{3.3}$$

(Recall that m is a function of L and k.) Thus the upper bound we derived depicts a simple exponential relationship between α and N/k. Next, we try to construct memory schemes of this type. A Common Lisp program using a modified depth-first search constructed memory schemes for various parameter values, whose α's came within 80% to 90% of the upper bound. These results are far from conclusive, however, since only a small portion of the parameter space was tested.

In evaluating the viability of this approach, its apparent optimality should be contrasted with two major weaknesses. First, this type of memory scheme is hard to construct computationally. It took our program several minutes of CPU time on a Symbolics 3600 to produce reasonable solutions for cases like $N = 200$, $k = 5$, $m = 1$, with an exponential increase in computing time for larger values of m. Second, if CCSMs are used as models of memory in naturally evolving systems (such as the brain), this approach places too great a burden on developmental mechanisms.

The importance of the bounded overlap approach lies mainly in its role as an upper bound for all possible memory schemes, subject to the simplifications made earlier. No scheme with guaranteed capacity ($P_{\text{ghost}} = 0$) is likely to yield a better scaling behavior than that of equation 3.3.

3.2 Random fixed size patterns (a stochastic approach)

Randomly produced memory schemes are easy to implement and are attractive because of their naturalness. However, if the patterns of two symbols coincide, the guaranteed capacity will be zero (storing one of these symbols will render the other a ghost). We therefore abandon the goal of guaranteeing a certain capacity, and instead establish a tolerance level for ghosts, P_{ghost}. For large enough memories, where stochastic behavior is more robust, we may expect reasonable capacity even with very small P_{ghost}.

In the first stochastic approach we analyze, patterns are randomly selected subsets of a fixed size L. Unlike in the previous approach, choosing k does not bound α. We may define as many symbols as we wish, although at the cost of increased probability of a ghost (or, alternatively, decreased capacity). The probability of a ghost appearing after k symbols have been stored is given by

$$P_{\text{ghost}}(N, L, k, \alpha) = 1 - \sum_{c=L}^{\min(N,kL)} T_{N,L}(k, c) \cdot \left[1 - \frac{\binom{C}{L}}{\binom{N}{L}}\right]^{\alpha - k}. \quad (3.4)$$

$T_{N,L}(k, c)$ is the probability that exactly c units will be active after k symbols have been stored. It is defined recursively by

$$\begin{aligned} &T_{N,L}(0, 0) = 1 \\ &T_{N,L}(k, c) = 0 \quad \text{for either } k = 0 \text{ and } c \neq 0, \text{ or } k > 0 \text{ and } c < L \\ &T_{N,L}(k, c) = \textstyle\sum_{a=0}^{L} T(k-1, c-a) \cdot \binom{N-(c-a)}{a} \cdot \binom{c-a}{L-a} / \binom{N}{L}. \end{aligned} \quad (3.5)$$

We have constructed various coarse-coded memories with random fixed-size receptive fields and measured their capacities. The experimental results show good agreement with the above equation.

The optimal pattern size for fixed values of N, k, and α can be determined by binary search on equation (3.4), since $P_{\text{ghost}}(L)$ has exactly one maximum in the interval $[1, N]$. However, this may be expensive for large N. A computational shortcut can be achieved by estimating the optimal L and searching in a small interval around it. A good initial estimate is derived by replacing the summation in equation (3.4) with a single term involving $E[c]$: the expected value of the number of active units after k symbols have been stored. The latter can be expressed as

$$E[c] = N \cdot \left[1 - (1 - L/N)^k\right].$$

The estimated L is the one that maximizes the expression

$$\binom{E[c]}{L} \Big/ \binom{N}{L}$$

An alternative formula, developed by Joseph Tebelskis, produces very good approximations to equation (3.4) and is much more efficient to compute. After storing k symbols in memory, the probability P_x that a single arbitrary symbol x has become a ghost is given by

$$P_x(N, L, k, \alpha) = \sum_{j=0}^{L} (-1)^j \binom{L}{j} \binom{N-j}{L}^k \Big/ \binom{N}{L}^k. \quad (3.6)$$

If we now assume that each symbol's P_x is independent of that of any other symbol, we obtain:

$$P_{\text{ghost}} \approx 1 - (1 - P_x)^{\alpha - k}. \quad (3.7)$$

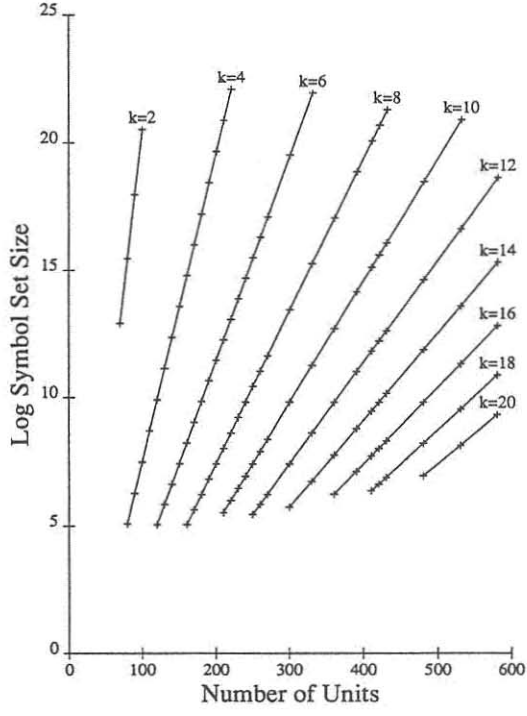

Figure 2: Log(symbol-set-size) vs. number of units for even capacity values (k) from 2 to 20. Probability of a *ghost* was set to 0.01. The optimal pattern size was used in each case.

This assumption of independence is not strictly true, but the relative error was less than 0.1% for the parameter ranges we considered, when P_{ghost} was no greater than 0.01.

We have constructed the two-dimensional table $T_{N,L}(k, c)$ for a wide range of (N, L) values ($70 \leq N \leq 1000$, $7 \leq L \leq 43$), and produced graphs of the relationships between N, k, α, and P_{ghost} for optimum pattern sizes, as determined by equation (3.4). A representative graph is shown in figure 2.

The results show an approximately exponential relationship between α and N/k. Thus, for a fixed number of symbols, the capacity is proportional to the number of units. Let α_{rfp} denote the maximum number of symbols attainable in memory schemes that use random fixed-size patterns. Then some typical relationships, derived from the data, are:

$$\begin{aligned} \alpha_{rfp}(P_{\text{ghost}} = 0.01) &\approx 0.0086 \cdot e^{0.468\frac{N}{k}} \\ \alpha_{rfp}(P_{\text{ghost}} = 0.001) &\approx 0.0008 \cdot e^{0.473\frac{N}{k}}. \end{aligned} \tag{3.8}$$

3.3 Random receptors (a stochastic approach)

A second stochastic approach is to have each unit assigned to each symbol with an independent fixed probability s. This method lends itself to easy mathematical analysis, resulting in a closed-form analytical solution.

After storing k symbols, the probability that a given unit is active is $1 - (1 - s)^k$ (independent of any other unit). For a given symbol to be a ghost, every unit must either be active or else not belong to that symbol's pattern. That will happen with a probability $\left[1 - s \cdot (1 - s)^k\right]^N$, and thus the probability of a ghost is

$$P_{\text{ghost}}(\alpha, N, k, s) = 1 - \left[1 - \left[1 - s \cdot (1 - s)^k\right]^N\right]^{\alpha - k}. \tag{3.9}$$

Assuming $P_{\text{ghost}} \ll 1$ and $k \ll \alpha$ (both hold in our case), the expression can be simplified to

$$P_{\text{ghost}}(\alpha, N, k, s) = \alpha \cdot \left[1 - s \cdot (1 - s)^k\right]^N,$$

from which α can be extracted:

$$\alpha_{rr}(N, k, s, P_{\text{ghost}}) = \frac{P_{\text{ghost}}}{\left[1 - s \cdot (1 - s)^k\right]^N}. \tag{3.10}$$

We can now optimize by finding the value of s that maximizes α, given any desired upper bound on the expected value of P_{ghost}. This is done straightforwardly by solving $\partial\alpha/\partial s = 0$. Note that $s \cdot N$ corresponds to L in the previous approach. The solution is $s = 1/(k + 1)$, which yields

$$\begin{aligned}\alpha_{rr}(s=\frac{1}{k+1}) &= P_{\text{ghost}}\cdot\left[1-\frac{1}{k+1}\left(1-\frac{1}{k+1}\right)^k\right]^{-N}\\ &= P_{\text{ghost}}\cdot\left[\frac{(k+1)^{k+1}}{(k+1)^{k+1}-k^k}\right]^N\\ &= P_{\text{ghost}}\cdot e^{N\log\frac{(k+1)^{k+1}}{(k+1)^{k+1}-k^k}}. \qquad (3.11)\end{aligned}$$

For large k we obtain

$$\alpha_{rr}(N,k,P_{\text{ghost}}) = P_{\text{ghost}}\cdot e^{\frac{N}{ek}} \approx P_{\text{ghost}}\cdot e^{0.368\frac{N}{k}}. \qquad (3.12)$$

A comparison of the results of the two stochastic approaches reveals an interesting similarity. For large k, with $P_{\text{ghost}} = 0.01$, the multiplicative factor of 0.0086 in equation (3.8) approximates P_{ghost} in equation (3.12), and similarly for $P_{\text{ghost}} = 0.001$. The coefficient in the exponent is larger in the case of fixed-size patterns. This is hardly surprising, since an exceptionally small pattern, which may be generated by the Random Receptors method, has a high probability of becoming a ghost, and conversely an exceptionally large pattern has a high probability of creating one. The advantage of fixed-length patterns, which was predicted in section 3.1, is now demonstrated analytically. According to the Law of Large Numbers, in the limit ($N, k, \alpha \to \infty$, with $k \ll N \ll \alpha$) the two methods are equivalent.

For Large k, the Bounded Overlap method and the Random Receptors method yield virtually identical exponents. Note, however, that the former guarantees its capacity, whereas the latter does not. Note also that α_{bo} is only an upper bound, derived solely for the parameter ranges we considered here.

It should be noted that the stochastic approaches we analyzed generate a family of memory schemes, with non-identical ghost-probabilities that depend on the particular patterns chosen. P_{ghost} in our formulas is therefore better understood as *an expected value*, averaged over the entire family.

3.4 Partitioned binary coding (a reference point)

The last memory scheme we analyze is not strictly distributed. Rather, it is somewhere in between a distributed and a localist representation, and is presented for comparison with the previous results. For a given number of units N and desired capacity k, the units are partitioned into k equal-size "slots", each consisting of N/k units (for simplicity we assume that k divides N). Each slot is capable of storing exactly one symbol.

The most efficient representation for all possible symbols that may be stored into a slot is to assign them binary codes, using the N/k units of each slot as bits. This would allow $2^{N/k}$ symbols to be represented. Using binary coding, however, will not give us the required capacity of 1 symbol, since

binary patterns subsume one another. For example, storing the code '10110' into one of the slots will cause the codes '10010', '10100' and '00010' (as well as several other codes) to become ghosts.

A possible solution is to use only half of the bits in each slot for a binary code, and set the other half to the binary complement of that code (we assume that N/k is even). This way, the codes are guaranteed not to subsume one another. Let α_{pbc} denote the number of symbols representable using a partitioned binary coding scheme. Then,

$$\alpha_{pbc} = 2^{N/2k} = e^{0.347\frac{N}{k}}. \tag{3.13}$$

Once again, α is exponential in N/k. The form of the result closely resembles the estimated upper bound on the Bounded Overlap method given in equation (3.3). There is also a strong resemblance to equations (3.8) and (11), except that the fractional multiplier in front of the exponential, corresponding to P_{ghost}, is missing. P_{ghost} is 0 for the Partitioned Binary Coding method, but this is enforced by dividing the memory into disjoint sets of units rather than adjusting the patterns to reduce overlap among symbols.

As mentioned previously, this memory scheme is not really distributed in the sense used in this paper, since there is no one pattern associated with a symbol. Instead, a symbol is represented by any one of a set of k patterns, each N/k bits long, corresponding to its appearance in one of the k slots. To check whether a symbol is present, all k slots must be examined. To store a new symbol in memory, one must scan the k slots until an empty one is found. Equation (3.13) should therefore be used only as a point of reference.

3.5 Comparison of results

Table 1 summarizes the results obtained for the four methods analyzed. Note that these results assume the use of optimal pattern sizes. The effect of using other pattern sizes can be derived from the analyses in the preceding sections.

Memory Scheme	Result
Bounded Overlap	$\alpha_{bo}(N,k) < e^{0.367\frac{N}{k}}$
Random Fixed-size Patterns	$\alpha_{rfp}(P_{\text{ghost}} = 0.01) \approx 0.0086 \cdot e^{0.468\frac{N}{k}}$ $\alpha_{rfp}(P_{\text{ghost}} = 0.001) \approx 0.0008 \cdot e^{0.473\frac{N}{k}}$
Random Receptors	$\alpha_{rr} = P_{\text{ghost}} \cdot e^{0.368\frac{N}{k}}$
Partitioned Binary Coding	$\alpha_{pbc} = e^{0.347\frac{N}{k}}$

Table 1: Summary of results for various memory schemes.

Some differences must be emphasized:

α_{bo} and α_{pbc} deal with guaranteed capacity, whereas α_{rfp} and α_{rr} are meaningful only for $P_{\text{ghost}} > 0$.

α_{bo} is only an upper bound.

α_{rfp} is based on numerical estimates.

α_{pbc} is based on a scheme which is not strictly coarse-coded.

The similar functional form of all the results, although not surprising, is aesthetically pleasing.

There is a simple linear relationship between the resources allocated to the system (N) and the capacity they yield (k). This is true no matter what the other parameters are. This result can be assumed to hold in all reasonably efficient memory schemes. Thus if we are given a working CCSM whose characteristics are known, perhaps through empirical measurements, *we can increase its capacity by any desired factor by simply increasing the number of units by that factor.* This may be a handy rule of thumb for designers and users of coarse-coded memories. Alternatively, if a working CCSM is observed to scale qualitatively worse than the models we investigated here, e.g. have a sublinear dependency of k on N, then in all likelihood the scheme being used is inherently inefficient.

Similarly, there is a linear tradeoff between the expressive power of the system (α, the number of symbols it can represent) and the tolerated probability of an error (P_{ghost}). Note that P_{ghost} is defined as the probability that *any* of the α symbols is a ghost, i.e. it already takes into account the size of the symbol set. The probability of a *given* symbol being a ghost is much smaller than P_{ghost}. Thus a second rule of thumb we derive is that *the probability of error can be reduced by a proportionate reduction in the symbol set size.*

Finally, all four parameters are related exponentially, through a coefficient which may vary from one memory scheme to another, but can be expected to remain within a rather narrow range. Since α/P_{ghost} is exponential in N/k, a third rule of thumb to emerge is that *the system's behavior is most sensitive to the ratio N/k, much more than to the ratio α/P_{ghost}.* For this reason it makes sense, when analyzing a memory scheme, to ignore the latter ratio and to evaluate the system's efficiency in terms of the number of bits required per stored symbol.

4. Economy of scale in coarse-coded memories

Coarse-coded symbol memories are characterized by a resource-sharing strategy. It is this pooling of resources that is responsible for the efficiency with which they handle sparse memories. It may be tempting to conclude that one large CCSM is in general more efficient than several (say m) smaller CCSMs providing the same functionality. We now discuss the conditions under which this is true. For simplicity's sake, we restrict our analysis to the case $m = 2$. We believe that the qualitative results apply to the general case as well.

4.1 Merging two CCSMs

Consider a system that requires two sparse memories. Each one must be able to represent α symbols (from disjoint alphabets) and to hold up to k of them in storage at any one time, while maintaining the probability of a ghost below P_{ghost}. Let us use a separate CCSM for each memory requirement and choos the Random Receptors memory scheme (see section 3.3). We assume that k is large enough so that equation (3.12) can be used. The number of units we need for both CCSMs together is

$$N_{2-\text{CCSMs}} = 2\,e\,k\,(\log\alpha - \log P_{\text{ghost}}). \tag{4.1}$$

If, on the other hand, we use one CCSM to provide the same functionality, we need a representation power of 2α symbols and a capacity of $2k$ symbols, while maintaining the same P_{ghost}. Using the same memory scheme, the number of units required for the single, "merged" CCSM is

$$\begin{aligned} N_{\text{merged}} &= e \cdot 2k \cdot (\log 2\alpha - \log P_{\text{ghost}}) \\ &= 2\,e\,k\,(\log\alpha + \log P_{\text{ghost}} + \log 2) \\ &= N_{2-\text{CCSMs}} + 2\,\log 2\,e\,k \\ &\approx N_{2-\text{CCSMs}} + 3.77k \end{aligned} \tag{4.2}$$

Thus the single, combined CCSM requires *more* resources than the two separate CCSMs.

4.2 Splitting a single CCSM

Consider now a system that requires a single memory with parameters α, k, P_{ghost}. Using a single CCSM and the Random Receptors memory scheme again, the number of units we need is

$$N_{1-\text{CCSM}} = e\,k\,(\log\alpha - \log P_{\text{ghost}}) \tag{4.3}$$

We now consider *splitting* our memory into two smaller CCSMs. We may do this by splitting the symbol set randomly into two halves, and accommodating each half using a separate CCSM. In this way, any k pre-selected symbols will be distributed binomially between the two smaller CCSMs. Using the same memory scheme and an as-yet-undetermined total number of units (N_{split}), the probability of a ghost in the split system is

$$\begin{aligned} &P_{\text{ghost}}(N_{\text{split}}, \alpha, k) = \\ &\quad 1 - \tfrac{1}{2^k}\textstyle\sum_{j=0}^{k}\binom{k}{j}[1 - P_{\text{ghost}}(\frac{N_{\text{split}}}{2}, \frac{\alpha}{2}, j)][1 - P_{\text{ghost}}(\frac{N_{\text{split}}}{2}, \frac{\alpha}{2}, k-j)]. \end{aligned} \tag{4.4}$$

We now require the probability of a ghost in the split system to be the same as that of the original system. Using this constraint, $N_{\text{smallsplit}}$ can be extracted numerically.

Figure 3 depicts both N_{split} and $N_{1-\text{CCSM}}$ as a function of the required capacity, for typical values of α and P_{ghost}. The single, undivided memory is significantly more efficient than the two-CCSM alternative, requiring 25–50% fewer units in the parameter range of the graph.

4.3 Comparison and analysis

We analyzed two scenarios. In the first, we considered merging two CCSMs into one, and proved it disadvantageous, requiring more units than the original system. In the second scenario, we considered splitting a single CCSM into two halves, and concluded similarly that such a split would not be beneficial, at least with regard to the resources needed. The seeming contradiction between the two results can be explained by careful analysis of the differences between them.

In the first case, the capacity requirements were specified separately for the two symbol sets. Each memory was required to be able to hold up to k symbols, and therefore the merged CCSM needed a capacity of $2k$ symbols. Note that this means that the merged CCSM will accommodate *more than k symbols from a single symbol set*, as long as the total number of stored symbols is less then $2k$. This "extra capacity" is not afforded by the two separate CCSMs. Thus it seems that "we got more than we bargained for" when we merged the two CCSMs into one. This added capacity helps us understand why the merged CCSM requires more units than the 2-CCSM system.

In the second case, there is only one symbol set, and hence only one capacity requirement. The usage profiles of the various symbols may in fact be correlated. For example, it is possible that a certain pair of symbols will never be stored together. If we knew of such a case we could use that fact to our advantage by, say, putting the two symbols into the same pool. Unfortunately, we don't know anything about the likely distribution of future storage demands, so the best we can do is split the system randomly, which results in a binomial distribution. Having no a priori knowledge about the intended use of the system, we cannot gain anything by splitting it into modules. The "economy of scale" argument mentioned before makes the non-split system more efficient, accounting for the data in figure 3.

The conclusion we draw from this analysis is that knowledge about the likely distribution of memory demands can be used to design a more efficient CCSM. However, when no such knowledge exists, merging all resources is likely to yield the most efficient system.

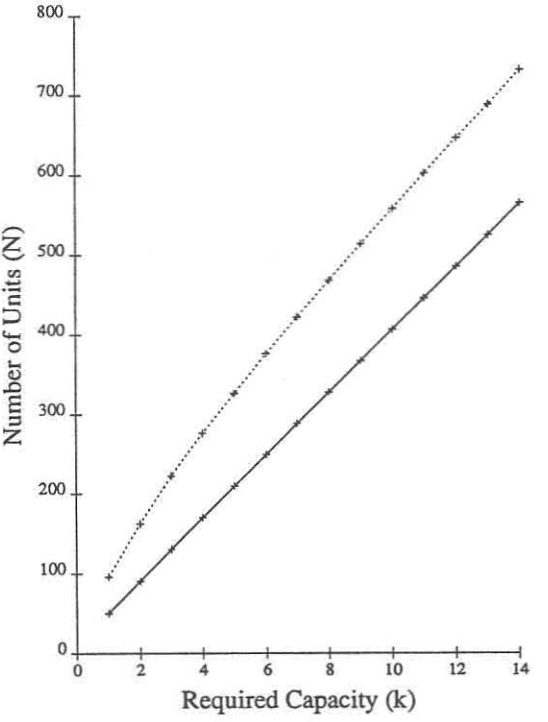

Figure 3: Economy of scale in CCSMs. Number of units (N) vs. the required Capacity (k) for original (solid line) and split memory (dotted line). Symbol-set size is 2000. Probability of a ghost is 0.001.

5. Structured symbol memories

The three distributed schemes we have studied all use unstructured patterns (as discussed in the introduction), the only constraint being that patterns are at least roughly the same size. In this section we discuss structured memories, beginning with an examination of the working memory of DCPS. We then show how the Wickelfeature representation used in the Rumelhart and McClelland verb learning model can be understood as a structured CCSM.

5.1 Measurement of DCPS

Imposing structure on the receptive fields of units, using any of the coarse coding schemes we have discussed, is likely to reduce the capacity somewhat. In order to quantify this effect, we measured the memory capacity of DCPS and compared the results with one of the theoretical models analyzed above.

There are 2000 units in the working memory of DCPS. Its symbols are triples of letters drawn from an alphabet of size 25. All possible combinations of letters are permitted, resulting in a three dimensional symbol space of size $\alpha = 25^3 = 15625$ triples. The receptive field of each unit is defined by the cartesian product of three randomly-chosen sets of six letters each. Thus each receptive field is a three dimensional subspace of form

$$\{\langle a, b, c\rangle \mid a \in A, b \in B, c \in C\},$$

where A, B and C are independent, randomly-chosen six letter sets. Units therefore have fixed-size receptive fields containing $6^3 = 216$ triples. (Each triple counts as one symbol.) DCPS requires structured receptive fields so that a distributed winner-take-all network, called a "bind space", can be used to decompose triples into their component letters.

The symbols in DCPS' working memory do not have fixed pattern sizes. The expected pattern size is $(6/25)^3 \cdot 2000 \approx 28$. Touretzky and Hinton manipulated the receptive fields as described in [4] to artificially reduce the variance from this mean. In the current implementation of DCPS, pattern sizes vary from 23 to 33, but most symbols have patterns containing 26 to 29 units; the standard deviation is only 1.5.

Figure 4 shows P_{ghost} as a function of k for the Random Receptors method as estimated by equation (11) and for DCPS (based on 10,000 trials, each consisting of storing randomly-generated triples until the first appearance of a ghost). N is 2000 and α is 15625. The two curves are quite close. Note that when P_{ghost} is 0.01, we observe an actual capacity of 48 symbols for DCPS[2] and an expected capacity of 51 symbols for the random receptors scheme. We thus conclude that for the parameter ranges discussed here, the structure in DCPS's fixed-size receptive fields (which have been manipulated to assure nearly fixed-size patterns) results in only a slight penalty relative to the random receptors approach.

[2]This measurement is based on a 100% visibility criterion, which we use throughout this paper. It therefore differs from previously reported values where lower visibility criteria were used [4].

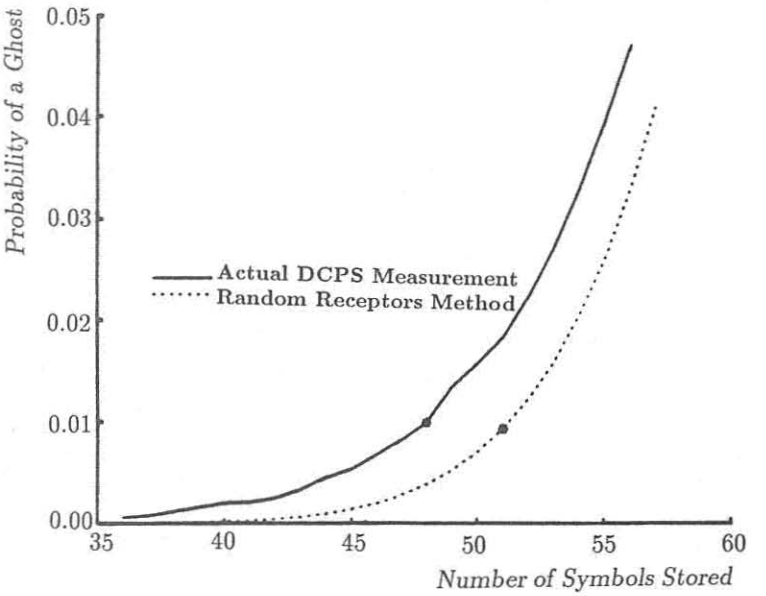

Figure 4: Probability of a ghost as a function of the number of symbols stored, measured for DCPS (solid line) and computed for the Random Receptors method (dotted line). The point on each line where P_{ghost} reaches 0.01 is marked by an asterisk.

Smolensky has recently shown that this form of structured, coarse-coded memory can be viewed as a special case of a tensor product representation [13]. In his formalism, each letter that appears in a DCPS receptive field table has an associated 2000-bit pattern. (Call the letters appearing in the first column A_1 through Y_1. The bit pattern for A_1 has a 1 in position i iff A_1 appears in the first column of the receptive field table of unit i. Similarly for the other letters, and for the other columns. A total of $(6/25) \cdot 2000 = 480$ bits will be on in each pattern.) Each of the 2000 working memory units then becomes an element on the major diagonal of a rank three tensor defined by the tensor product of three 2000-bit vectors.

The tensor product approach is interesting because it suggests alternatives to DCPS-style clause spaces for retrieving elements from memory. On the other hand, CCSMs use their units more efficiently because they have far less redundancy. Measurement of efficiency in practice is complicated by the retrieval issue; for self-addressing, which is the simplest retrieval method for tensor product representations and requires taking the inner product of a rank three tensor with a rank two tensor, the number of items that can be stored before retrieval fails is difficult to derive analytically. Empirical measurements are currently in progress.

5.2 Wickelfeature representations

The Rumelhart and McClelland verb learning model [2] uses an interesting coarse-coded representation for triples of phonemes, or "Wickelphones". The Wickelphone $_{\mathrm{k}}\mathbf{A}_{\mathrm{m}}$ denotes a phoneme /A/ whose left context is /k/ and whose right context is /m/. Words are encoded as sets of Wickelphones; beginning and ending word boundaries are marked by #, which appears as the left context of the first Wickelphone and the right context of the last Wickelphone making up the word.

Each phoneme is described by an eleven-bit vector with either four bits on (for consonants and vowels) or one bit on (for word boundary markers). Each of these bits stands for some phonetic feature, such as Front, Back, Stop, Nasal, Voiced, and so on. The encoding provides for 35 distinguishable phonemes. Each of the 460 Wickelfeature units in the coarse coded representation stands for a conjunction of three of these phonetic features, one from each of the three phonemes making up a Wickelphone. For example, one of the units that participates in the representation of $_{\mathrm{k}}\mathbf{A}_{\mathrm{m}}$ is active for Wickelphones whose left context is a stop like /k/, whose central phoneme is voiced (for consonants) or long (for vowels) like /A/, and whose right context is a nasal like /m/.

If phonetic features are microfeatures, then Wickelfeatures are three-way conjunctions of microfeatures. One can look at a single active Wickelfeature unit and learn something useful (for purposes of the verb learning task) about the properties of the Wickelphone it encodes. In contrast, the individual units in a structured CCSM such as DCPS have no simple interpretation, because the letters that make up a receptive field table are not grouped together by shared microfeatures.

Consider the [Stop, Voiced/Long, Nasal] Wickelfeature unit described previously. It is defined by these three phonetic features, but it can also be described in terms of a cartesian product receptive field table. The receptive field table contains, in the first column, all the stops; in the second column, all phonemes that are either voiced consonants or long vowels; and in the third column, all the nasals.[3]

When individual units are described using Wickelfeatures they appear uniform, but when viewed as receptive fields of Wickelphones we see that there is substantial variation from unit to unit. For example, Vowel is a feature of twelve distinct phonemes; Interrupted is a feature of nine phonemes; Stop/Fricative/High is a feature of eighteen phonemes, while Word Boundary is a feature of only one phoneme. So the Wickelfeature unit [Word Boundary, Interrupted, Vowel] codes for $1 \cdot 9 \cdot 12 = 108$ Wickelphones, while the unit [Vowel, Stop, Vowel] codes for 972. Since the encoding provides for 35 distinguishable phonemes plus the word boundary marker, there are $35^3 + 2 \cdot 35^2$

[3]Actually the first column contains stops, fricatives, and high vowels, because the mutually exlcusive features Stop, Fricative, and High are all represented by the same bit in the eleven-bit encoding. There is really a Stop/Fricative/High feature rather than three separate ones. Likewise, the third column contains not just nasals, but also liquids, semivowels, and low vowels.

or 45325 Wickelphones in all. The varying coarseness of the units' receptive fields makes it difficult to estimate the capacity of this memory, but it appears to be adequate for the intended application.

The lesson to be drawn from the Wickelfeature example is that coarse coding is not incompatible with a conjunctive microfeature representation. Although one gives up a certain amount of capacity in return for the additional structure, useful properties may result. The verb learning model showed generalization to novel verbs precisely because of the overlapping representations of phonetically similar Wickelphones. Rumelhart and McClelland further blurred the representation (by introducing noise into the receptive fields) to enhance this generalization effect, which also reduced the amount of training required.

6. Constructing coarse-coded memories

Returning to the main topic of this paper, which is CCSMs with maximal efficiency rather than domain-specific internal structure, we note that the three distributed schemes studied here have similar properties in the limit. However, when constructing actual models that use these representations, one generally cannot afford to simulate more than a few thousand units. This is too many units to generate optimal receptive fields as defined by the Bounded Overlap method, because of the exponential search involved. On the other hand, it is too few units to permit reliance on either of the two simple stochastic schemes. Unless the memory parameters are large, the variance in pattern size (for the Random Receptors method) and in the overlap between patterns (for both stochastic methods) may be large enough to interfere with the operation of the generated memory.

For practical application of CCSMs, we present an algorithm for heuristically generating good fixed-size receptive fields. The algorithm guarantees a uniform pattern size (provided that N divides α), and attempts to minimize the overlap between patterns. For a given receptive field size F, the pattern size will be $L = N \cdot F/\alpha$, and the expected overlap between two patterns is $O_E = L \cdot (F-1)/(\alpha - 1)$.

1. Initialize. Let $U[1..N]$ be an array of sets defining the receptive fields of the N units. Initialize U to sets of consecutive symbols in the infinitely repeating sequence $\mathcal{S} = \{S_1, \ldots, S_\alpha, S_1, \ldots\}$, so that $U[1] = \{\mathcal{S}_1, \ldots, \mathcal{S}_F\}$, $U[2] = \{\mathcal{S}_{F+1}, \ldots, \mathcal{S}_{2F}\}$, and so on. This step guarantees equal size patterns and equal size receptive fields.

2. Shuffle. Pick two distinct units x and y at random. Pick two symbols S_i and S_j such that $S_i \in U[x]$ and $S_j \in U[y]$, and $S_i \notin U[y]$ and $S_j \notin U[x]$. Swap S_i and S_j in the two receptive field tables. Repeat this step some multiple of $N \cdot F$ times, to assure adequate shuffling of all receptive fields.

3. Compute the variance. Let $C[1..\alpha; 1..\alpha]$ be an array of integers in $[0, L]$. For each pair of symbols S_i, S_j, let $C[i, j]$ be the number of units

whose receptive fields contain both symbols. Note that $C[i,i] = L$ for $1 \leq i \leq \alpha$. Ideally, $C[i,j] = O_E$ for $1 \leq i,j \leq \alpha$ when $i \neq j$. Define the variance V to be the difference between the expected and actual overlap for all pairs of symbols: $V = \frac{1}{2}\sum_{i \neq j}(C[i,j] - O_E)^2$.

4. Reduce the variance. Pick two distinct units x and y at random. Pick two symbols S_i and S_j such that $S_i \in U[x]$ and $S_j \in U[y]$, and $S_i \notin U[y]$ and $S_j \notin U[x]$. Swap S_i and S_j in the two receptive field tables, and update the co-occurrence matrix C. (The update will affect up to $4(F-1)$ of the α^2 entries. Half will be decremented, the other half incremented.) Let V' be the new variance computed after C is updated. If $V' \leq V$ accept the swap; otherwise undo it. Repeat this step until the variance has been reduced to an acceptable level.

The last step of the algorithm assures a monotonic decrease in variance over time, so one can stop at any point when the current state is "good enough." In contrast, the depth first search algorithm we used to generate optimal fields for the Bounded Overlap method must run to completion in order to obtain a full set of N receptive fields. Our heuristic algorithm does not guarantee optimal fields because the search can get trapped in local minima, but experience with models such as DCPS and BoltzCONS suggests that in practice this is not a serious problem.

The algorithm is also applicable to structured memories, where one may swap a letter in any column of one unit's receptive field table with a letter in the corresponding column in some other unit's table. The major change to the algorithm is in the computation of which symbols are affected by a swap: in DCPS, replacing one letter in one column of the table causes the replacement of 36 symbols, due to the cartesian product that generates the receptive field. Fixed pattern sizes are difficult to obtain when cartesian products are involved, but good approximations are possible. Instead of computing variance based on the co-occurrence rate of pairs of symbols (which will depend on the number of components they have in common), the last step of the algorithm should manipulate the receptive field tables to minimize the variance in pattern size.

Acknowledgments

We thank Geoffrey Hinton, Noga Alon and Victor Wei for helpful comments, and Joseph Tebelskis for sharing with us his formula for approximating P_{ghost} in the case of fixed pattern sizes.

This work was supported by National Science Foundation grants IST-8516330 and EET-8716324, and by the Office of Naval Research under contract number N00014-86-K-0678. The first author was supported by a National Science Foundation graduate fellowship.

References

[1] G. H. Hinton, J. L. McClelland, and D. E. Rumelhart, "Distributed Representations", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 1*, D. E. Rumelhart and J. L. McClelland, eds. (MIT press, 1986).

[2] D. E. Rumelhart and J. L. McClelland, "On Learning the Past Tenses of English Verbs", in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, volume 2*, J. L. McClelland and D. E. Rumelhart, eds. (MIT press, 1986).

[3] D. S. Touretzky and G. E. Hinton, "Symbols Among the Neurons: Details of a Connectionist Inference Architecture", *Proceedings of IJCAI-85*, held at Los Angeles, CA (1985) 238–243.

[4] D. S. Touretzky and G. E. Hinton, "A Distributed Connectionist Production System", (to be published in *Cognitive Science* **12**(3) (1988)).

[5] D. S. Touretzky, "BoltzCONS: Reconciling Connectionism With the Recursive Nature of Stacks and Trees.", *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA (1986) 522–530.

[6] D. S. Touretzky, "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*, Atlanta, GA (1986) 12–16.

[7] M. F. St. John and J. L. McClelland, "Reconstructive memory for sentences: a PDP approach", *Proceedings of the Ohio University Inference Conference* (1986).

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

[9] D. H. Ballard, "Cortical connections and parallel processing: structure and function", *Behavioral and Brain Sciences*, **9**(1) (1986).

[10] J. Jullins, "Value cell encoding strategies", Technical report TR-165 (1985), Computer Science Department, University of Rochester, Rochester, NY.

[11] J. J. Hopfield, (1982) "Neural Networks and physical systems with emergent collective computational abilities", *Proceedings of the National Academy of Sciences, USA*, **79** (1982) 2554–2558.

[12] F. J. Macwilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, (North-Holland, 1978).

[13] P. Smolensky, "On Variable Binding and the Representation of Symbolic Structures in Connectionist Systems", Technical report CU-CS-355-87, Department of Computer Science, University of Colorado at Boulder.