Sparse Distributed Memory Using Rank-Order Neural Codes

Stephen B. Furber, *Fellow, IEEE*, Gavin Brown, Joy Bose, John Michael Cumpstey, Peter Marshall, and Jonathan L. Shapiro

***Abstract*—A variant of a sparse distributed memory (SDM) is shown to have the capability of storing and recalling patterns containing rank-order information. These are patterns where information is encoded not only in the subset of neuron outputs that fire, but also in the order in which that subset fires. This is an interesting companion to several recent works in the neuroscience literature, showing that human memories may be stored in terms of neural spike timings. In our model, the ordering is stored in static synaptic weights using a Hebbian single-shot learning algorithm, and can be reliably recovered whenever the associated input is supplied. It is shown that the memory can operate using only unipolar binary connections throughout. The behavior of the memory under noisy input conditions is also investigated. It is shown that the memory is capable of improving the quality of the data that passes through it. That is, under appropriate conditions the output retrieved from the memory is less noisy than the input used to retrieve it. Thus, this memory architecture could be used as a component in a complex system with stable noise properties and, we argue, it can be implemented using spiking neurons.**

***Index Terms*—Associative memory, neural networks (NNs), rank-order codes, sparse distributed memory (SDM), spiking neurons.**

I. Introduction

THE sparse distributed memory (SDM) [1]–[3] is an interesting form of *associative memory* [4], popular in both the computer science [5] and psychology [6] literature as it represents both an interesting information storage device and a plausible mathematical model of human long-term memory. The SDM associates two binary vectors x and y by projecting x into a very high-dimensional intermediate vector w, and then associating $x \rightarrow w$ and $w \rightarrow y$. The principle is founded on the distribution of points in high-dimensional spaces, such that any given point is relatively far from most of the rest of the space and from other points of interest [1]. In this way, errors can be made in the reproduction of w in the intermediate space without significant errors in the reproduction of the final memory y in the output space. The details of this will be expanded upon later.

A. Motivation

Several authors have investigated how an SDM can be used to store and recall *temporal sequences* of memories; this is conventionally achieved in a *heteroassociative* manner, feeding successive outputs back as the next input. A recent example is the work by Rehn and Sommer [7], who employ a Willshaw memory [4] to store associations between successive sparse volleys of spikes to encode a visual image. Alternatively, an *autoassociative* memory may be used to learn and store relative spike times, as in the hippocampal model by Lengyel *et al.* [8] where neurons learn to fire in phase patterns relative to the background theta-wave rhythm.

The original SDM was proposed by Kanerva as a model of human long-term memory; however, since this seminal work, much has come to light on the coding of human memories in terms of *temporal properties*, i.e., precisely *when* a neuron fires relative to others—the *rank order* of the neurons [9], [10]. In this paper, we investigate the viability of using the rank order of firing of a group of neurons to represent information, and then storing and recalling the firing sequence in an SDM framework. The rank ordering is stored as a *single memory*, as opposed to using the feedback schemes described previously, and the representation of the memories when stored is purely binary. An attractive feature of rank-order encoding is its potentially high information content compared with binary and unordered N-of-M encoding.

This paper is part of an ongoing investigation into the feasibility of building a large-scale hardware system for modeling very large numbers (up to a billion) of spiking neurons, perhaps using the *leaky integrate and fire* model [11] and employing rank-order codes. In this paper, we will not focus on the implementation of the SDM using spiking neurons, though we will offer pointers to such an implementation at the end of the paper.

B. Outline

In Section II, we give a brief review of Kanerva's SDM and related models. In Section III, we introduce our rank-order coded SDM model, and present an analysis of its performance in Section IV. In Section V, we look at the way the performance of the memory depends on design parameters and scale, and in Section VI, the memory performance with noisy inputs is investigated. We discuss possible ways to implement the memory using spiking neurons in Section VII, and present our conclusions in Section VIII.

Manuscript received January 20, 2006; revised September 15, 2006. This work was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) Advanced Processor Technologies Portfolio Partnership Award and by Cogniscience Limited. S. B. Furber is the recipient of a Royal Society Wolfson Research Merit Award.

S. B. Furber, G. Brown, J. Bose, J. M. Cumpstey, and J. L. Shapiro are with the School of Computer Science, the University of Manchester, Manchester M13 9PL, U.K. (e-mail: steve.furber@manchester.ac.uk).

P. Marshall was with the School of Computer Science, the University of Manchester, Manchester M13 9PL, U.K. He is now a freelance consultant working in the consumer electronics industry (e-mail: pete@goteck.co.uk).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNN.2006.890804

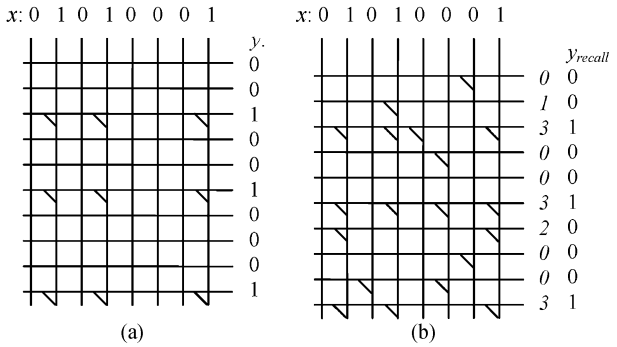


Fig. 1. (a) CMM. The markers at the wire intersections show the result of a single write operation, associating the shown input (x) and output (y) binary vectors. (b) CMM showing a read operation. The markers lying on active input lines are summed, and thresholded (in this case, the threshold is 3).

II. SDM and Related Models

In this section, we review the SDM framework, introduce notation that we will later expand for our own model, and set our work in relation to the existing literature.

There are two ways to understand an SDM, first as two grids of intersecting wires, and second in a pure algebraic framework. Here, we will use both interpretations as this serves to provide a better exposition of the ideas. The grids are correlation matrix memories (CMM) [5] and an SDM is made up of two of these. A CMM has *input* and *output* wires arranged in a grid. The task is to associate two binary patterns x and y placed on its input and output wires. If a wire has a bit set to 1, that wire is said to be *active*. For a write operation [see Fig. 1(a)] to the CMM, a marker is placed wherever two active wires cross. For a read operation [see Fig. 1(b)], when a new x is fed in, any markers that lie on active input wires are summed and a threshold is applied—any that reach/exceed the threshold have their output bits set to 1, otherwise 0. Fig. 1(b) shows the original memory was recalled in spite of the noise in the output (as a result of other memories having been written). It has been shown that at asymptotically large sizes and with sparse coding such memories can achieve a capacity of 0.7 bits of information per binary synapse [12].

As mentioned, an SDM uses two CMMs, one of which is fixed and the other is modified as data is loaded into the SDM. The first CMM uses x to recall a high-dimensional vector w which is used to read data out of the second CMM. The first memory, which we call the *address decoder memory* (because of the obvious analogy with a conventional computer random access memory), associates $x \rightarrow w$. The second, which we call the *data memory*, associates $w \rightarrow y$. In Kanerva's original SDM [1], the data memory used up/down counters at the wire intersections; however, here, we restrict ourselves to discussing the special case of binary connections. A write process consists of associating x to w, then associating w to y. A read process is simply the presentation of an x, resulting in a *word-line vector* w, which is then used to read a y value out of the data memory.

The same model can be described algebraically. The two CMMs form an association between the input $x \in \{0,1\}^M$ and the output $y \in \{0,1\}^M$ via the intermediate vector $w \in \{0,1\}^W$, where the intermediate dimension $W \gg M$. For example, an input could be $x = \{1,0,0,1,1,0,0,0,0,0\}^T$. Note that x is a column vector. This is a 3-of-10 code, i.e., $N = 3$ bits are on, from the total of $M = 10$. The address decoder matrix A is a $W \times M$ binary matrix, where W is the number of *address decoders*. Each row (decoder) in A has a fixed number of bits set a chosen randomly from the total M. We note that the SDM address matrix has itself been a focus of research, however, here it is not of concern to us, and we default to a random initialization.

A *read operation* from the memory begins with the generation of the high-dimensional *word-line vector* w. This is given by

$$w := \theta(Ax, t_A). \tag{1}$$

This is the inner (dot) product of x with each row of A, producing a column vector, thresholded by the Heaviside function θ, setting each element to 1 if it exceeds t_A and to 0 otherwise; this generates the binary column vector w. The data memory D is again a $W \times M$ binary matrix, though it is initially empty. The recall operation from D is

$$y := \theta(D^T w, t_D) \tag{2}$$

and the result y can be evaluated for its fidelity by its Hamming distance from the value originally associated with the input vector x. If we know that the final recalled value y has a definite number of bits set l, recall could also be implemented as

$$y := L(D^T w, l) \tag{3}$$

where L is the l-max function (analogous to k-winner-take-all [13]), setting the largest l elements to 1 and setting all other elements to 0. The recall of the word-line vector could also be performed with the l-max function if required.

A *write operation*, which associates two such vectors x and y, begins again with the recall of a word-line vector as in (1). Then, the contents of the data memory matrix D are updated by

$$D := D \oplus wy^T \tag{4}$$

where $\oplus$ is an update operator; this is also known as the *Hebbian* or *outer product* learning rule. The update operator in Kanerva's original SDM was simple addition, in which the word vector w was represented in binary unipolar $\{0,1\}$ form and the data y was represented in binary bipolar $\{-1,+1\}$ form. As mentioned, each connection in D required an up/down counter, and the precision could saturate at a small number of bits without any degradation of performance in practice. Willshaw's associative memory [4] and Austin's advanced distributed associative memory (ADAM) architecture [14] used logical OR as the update, the latter also using an input preprocessing step called *n-tupling*. We will introduce a novel threshold and update operator later; these form the cornerstone of the model presented in this paper.

III. Rank-Order Coded SDM Model

We will now describe the operation of an SDM model that is capable of storing and recalling rank-order significance vectors

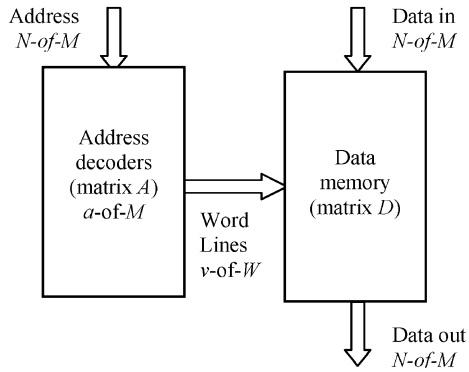


Fig. 2. Structure of an SDM that uses N-of-M codes. The address decoder has W outputs that each receive inputs from a randomly selected a of the M input address elements. For each input address, the v address decoder neurons with the highest activation fire. The data memory is a fully connected $W \times M$ correlation matrix memory.

with good accuracy, the accuracy achieved depending on the memory configuration and loading in a predictable way.

In order to build an SDM based upon rank-order codes we can start from the structure of the SDM as illustrated in Fig. 2. This structure is applicable to SDMs using unordered [15] or ordered N-of-M codes. For a rank-order coded SDM the address decoder layer should be sensitized to the rank order of the input address and it should generate a rank-order output code on the word lines (this term is taken from its analogous use in conventional digital memories; *word lines* are the outputs from the fixed address decoder logic that each activate a particular word in the modifiable data array). The data memory layer should be sensitive to the rank order of both the word-line inputs and the input data, in order to store and recover the rank order of the data.

In the course of describing the measures taken to achieve rank-order sensitivity we will demonstrate two results that we found surprising when we first encountered them.

— A simple binary-weighted address decoder can be configured to have the required input rank-order sensitivity.

— A simple binary-weighted data memory can be employed to capture and recover the rank order of the stored data.

Thus, the rank-order coded SDM can employ a structure with a very similar cost to the unordered N-of-M SDM, and the additional information content in the rank order can be stored and recovered at very little additional cost.

A. Representing a Rank-Order Code

There are several alternative ways to represent the rank-order firing sequence of a set of neurons. One way to represent a rank-order code is as an ordered list of N indices, each index having a value in the range 1 to M and no two indices having the same value. The first list element represents the first firing neuron; the last element represents the Nth firing neuron. For example, the rank-ordered 3-of-6 code $\{4, 3, 1\}$ is used to represent a group of six neurons in which neuron number 4 fires first, neuron 3 fires second and neuron 1 fires last.

The ordered list representation can be expanded into a fully occupied *rank vector* of size M where N elements are set to the relevant rank and (M-N) are 0. In the previous example, the rank vector is {third, 0, second, first, 0, 0}. More useful than this representation is a similar rank vector where the elements represent the *significance* of each firing neuron. The concept of significance reflects the assumption in a rank-order model that some positions in the order are more important than others. Typically, the significance associated with a rank-order input reduces monotonically (but not necessarily linearly) as more neurons fire, hence the first firing neuron is the most significant. In general, we can associate a numerical significance σ_i with the ith firing neuron and employ the *significance vector*

$$x = \{\sigma_3, 0, \sigma_2, \sigma_1, 0, 0\} \tag{5}$$

to represent the rank-order code. We will generally assume that the significance vector has been normalized, so that

$$\sum_{i=1}^{N} \sigma_i^2 = 1. \tag{6}$$

An example of this is $\boldsymbol{x} = \{0.81, 0, 0.9, 1, 0, 0\}$, which is then scaled to unit length, yielding the normalized significance vector $\boldsymbol{x}' = \{0.52, 0, 0.57, 0.64, 0, 0\}$.

Thus, all significance vectors lie on the surface of the first quadrant of the unit sphere in an M-dimensional space, though their distribution on this surface is far from even—each vector represents a different permutation of the same set of coordinates. We will employ a geometric significance function where for all i, we have $\sigma_{i+1}/\sigma_i = \sigma$, where σ is the *geometric significance ratio* and the ordered significance values $\{\sigma_1, \sigma_2, \sigma_3, \ldots\}$ form a geometric series. Throughout the rest of this paper σ will take the value 0.9 except where otherwise stated.

The normalized significance vector representation turns out to be extremely useful in *comparing* two codes—we will describe the issues surrounding this in Section III-B.

B. Comparing Two Rank-Order Codes

Comparing two unordered N-of-M codes is straightforward—the number of bits that differ between the two codes is simply the Hamming distance between them. It is therefore easy to judge the quality of a recovered N-of-M code by calculating the Hamming distance between it and the original code stored.

Comparing rank-order codes is potentially more complex. Here, errors may arise because of misplaced components as with N-of-M codes, but in addition there can be errors in the order of firing. For example, if the correct code is $\{0.52, 0, 0.57, 0.64, 0, 0\}$ an incorrect code with a misplaced component is $\{0, 0.52, 0.57, 0.64, 0, 0\}$ and an example of an incorrect code with an order error is $\{0.57, 0, 0.52, 0.64, 0, 0\}$. Both classes of error are captured if the measure of similarity of two rank-order codes is based on the scalar (dot) product of their respective (normalized) significance vectors. If $\boldsymbol{x}$ and $\boldsymbol{x}$' are the significance vectors of two rank-order codes that we wish to compare for equality, for example to establish whether or not we have recovered a rank-order code from a memory without error, we can use the test

$$x \cdot x' = 1 \Rightarrow x = x'. \tag{7}$$

Similarly, if we wish to allow for a certain level of error in the system, we can employ a weaker criterion by setting a threshold t and using the test

$$x \cdot x' > t \Rightarrow x \approx x'. \tag{8}$$

The dot-product threshold test is equivalent to using the Euclidean distance between the significance vectors in the M-dimensional space they occupy as a similarity metric, and for small distances this is very close to measuring their separation on the surface of the unit sphere in that space.

If we employ the threshold criterion to allow for errors, then we are mapping a number of rank-order codes together, thereby reducing the number of codes we view as distinct and the effective information content of each code. This is discussed in more detail in Section III-C.

It is of interest to note that, in the case of unordered N-of-M codes, the nonnormalized scalar product is linearly related to the Hamming distance $H(x;x')$. The relationship is

$$x \cdot x' = N - H(x;x')/2 \tag{9}$$

which may be explained by observing that the number of jointly active bits in x and x' is equal to the total number of active bits (N) less the number of bits that are active in x but not in x' $(H/2)$.

It may be useful to get a feel for the effect of different sorts of coding errors on the scalar product. Here, we illustrate the issues using an 11-of-256 rank-order code, which is the code we use throughout this paper. The choice of $N = 11$, $M = 256$ is not intended to be optimal, and is simply illustrative. It also helps comparisons of this paper with our previous work on unordered N-of-M codes, where we also used 11-of-256 codes [15]. The memory is entirely functional beyond $N = 11$; several experiments were performed with larger N. The limiting factor is the calculation of the information efficiency, which we describe in Section IV-A. Our current algorithm is computationally intractable for values of N much greater than 11.

Using an 11-of-256 rank-order code with a geometric significance ratio of 0.9, we observe the following.

- If the population of firing neurons is preserved but the order information is lost, the expected scalar product is $(\sum_{i=0}^{10}(0.9)^i)^2/11.\sum_{i=0}^{10}(0.9)^{2i} = 0.902$. Thus, any result greater than 0.9 has preserved the firing order, at least in part.
- If the last neuron in the rank order is lost but the order of the others is preserved, the scalar product is 0.974. Any result greater than this must have preserved the population information in its entirety.
- An error in the order of firing of the two least-significant neurons gives a scalar product of 0.9997; an error in the order of firing of the two most significant neurons gives a scalar product of 0.998. Local errors of order are therefore minor compared to population errors or significant loss of order.

We have illustrated that measuring the "distance" between two rank-order codes is not as simple as with simple binary codes, and that a dot product between significance vectors can be a useful measure. In Section IV-A, we will consider an information theoretic view and characterize the Shannon information content contained in the different coding schemes.

C. Rank-Order Address Decoder Matrix

A problem noted earlier with the unordered N-of-M address decoder was that it was not possible to control the precise number of address decoders that fire [15]. This is because the Hamming distance distribution is very coarsely stepped, leading to a large and variable number of address decoders sharing the same Hamming distance at the threshold level used to discriminate those that are active. As there is no way to select a subset of this group either all or none must be active, so significant variation in the number of firing address decoders must be accepted, with detrimental consequences for the efficiency of the memory. Rank-order codes give a much smoother range of dot-product values, enabling the number of address decoders that fire to be controlled more precisely.

Earlier work on neurons sensitized to rank order [9] required the neurons to have connection weights that can take a range of values. This has a higher cost than the address decoder in the N-of-M SDM which employs binary address decoder weights. However, although an *individual* neuron requires multivalued weights in order to be tuned to a particular input rank-order code, the same is not true of a *population* of neurons. This is because each member of the population samples from the input set independently.

If the firing order of two address inputs with significances σ_a and σ_b is reversed this will not be detected by a binary address decoder neuron that samples both inputs since $\sigma_a + \sigma_b = \sigma_b + \sigma_a$. However, it *will* affect the relative activation levels of address decoder neurons that sample only one of these two inputs, and their activation relative to a neuron that samples both. Since far more neurons sample one than sample both this is sufficient for the address decoder population to display strong sensitivity to the input rank order.

For example, if there are $W = 4096$ 11-of-256 binary address decoder neurons each sampling from an 11-of-256 rank-order input address, on average, 338 decoders will sample only one of any particular pair of inputs and seven decoders will sample both. Only those decoders that receive among the highest total activations will fire, and of the top 50, on average, just over 19 sample only one of the pair of inputs and two sample them both.

Thus, address rank-order sensitivity can be achieved with a binary-weighted address decoder matrix with only minor ambiguities resulting from the use of binary rather than multivalued weights.

D. Rank-Order Data Memory Matrix

The unordered N-of-M SDM uses single-bit connection weights between address decoder neurons and data neurons, and is able to store and retrieve unordered N-of-M data words with a performance similar to the correlation matrix memory that forms the basis of the data memory layer.

In order to store rank-ordered N-of-M codes, it is natural to consider using multivalued connection weights. When an active address decoder connects to an active data input, an obvious approach is to set the connection weight between them

to a function of both the rank of the firing address decoder and the rank of the associated data input, for example the product of the two significance values. If a subsequent write operation activates the same address decoder and data input pair, a decision must be taken whether and how the weight should be updated. One possibility is to use a function that selects the maximum of the existing weight and the new weight value, leading to weights displaying a monotonic increasing behavior over time.

However, as with the address decoder, a data memory that can store a range of different weight values will have higher cost than the binary memory used in the N-of-M SDM. Therefore, we sought an algorithm that allows rank-order codes to be stored in a binary data memory.

Clearly, if rank-order information is to be recovered from a binary memory, the order must be represented somehow in the way a data value is stored. The technique used here is to write that data into the memory in a way that causes the number of memory bits that are set to depend on the rank of the data bit. Thus, if the address decoder activates v word lines, the most significant data bit will be represented by $v1s$, the second data bit by $(v-s)1s$, the third by $(v-2s)1s$, and so on. s is the *skew*—the difference in the number of $1s$ set in data lines of adjacent significances. If both input data and word-line codes use the same geometric significance and the skew $s = 1$ this can be achieved by a weight update as follows:

$$t_D = \sigma_w^{\min} \cdot \sigma_y^{\max} \tag{10}$$

$$\text{if } \sigma_i \cdot \sigma_j \geq t_D \quad \text{then } D_{ij} := 1 \tag{11}$$

where σ_i is the significance of the word line, σ_j is the significance of the data bit, $\sigma_w^{\min}$ is the significance of the least-significant active word line, $\sigma_y^{\max}$ is the significance of the most significant active data bit, and D_{ij} is the strength of the synapse connecting this word line to the data neuron in question. Note that here we assume that the number of active word lines is greater than the number of active data lines.

For skew $s > 1$, we can achieve the desired pattern of written $1s$ by setting a different significance ratio σ_W for the word lines. It can be shown that, for skew s, we must have

$$\sigma_W = \sigma_D^{1/s} \tag{12}$$

(for example, we will use $\sigma_D = 0.9$ and $\sigma_W = 0.9655$ to give a skew of 3 in Section VI) and then apply (10) and (11) as before. If we imagine for a moment that the word lines and data bits are reordered so that the most significant values are to the top-left of the data memory and then decreasing significance moves us right across the data bits and down across the word lines, then the effect of a write operation is to set a trapezoidal area of the data memory to 1 (where, in the equivalent N-of-M SDM, we would have set the circumscribing rectangular area to 1). This is illustrated in Fig. 3.

E. Reading and Writing Rank-Order Memories

Both reading and writing operations start with the recall of a word-line significance vector w, which is associated with an input address x (also in significance vector form), through the address decoder matrix A

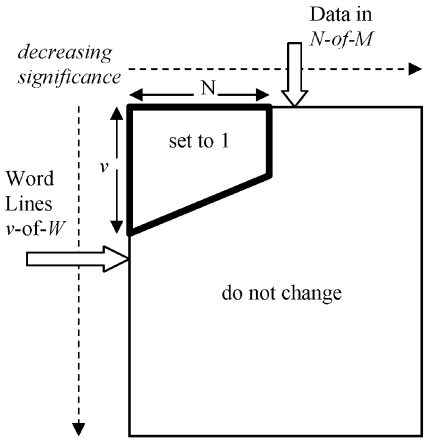


Fig. 3. Write operation in the data store matrix. If the word-line inputs are sorted so that the most significant is at the top, and the data input lines sorted so that the most significant is at the left, then a write operation is performed by setting all the data store locations within the trapezoid to one (whatever the value was before the write operation). The slope on the bottom edge of the trapezoid is the means of recording the order of the data inputs.

$$w := f(L'(Ax, 23)). \tag{13}$$

The function L' is the l-max function described earlier except that here it retains the values of the top $v = 23$ elements rather than replacing them with 1. The function f is a novel component of our model. It accepts a vector and replaces the nonzero elements (here, there are 23) with their corresponding significance values. Thus, the largest element is replaced with σ^0, the second largest with σ^1, the third largest by σ^2, and so on, until the 23rd element is replaced by σ^{22}. Then, the vector is normalized to unit length. The value σ is the geometric significance ratio introduced earlier.

Once the word-line vector has been recalled the read operation proceeds as follows:

$$y := f(L'(D^T w, 11)). \tag{14}$$

This comprises taking the dot product of w with each column in D, and then, applying the L' and f operators as before to yield an 11-of-256 significance (column) vector. The fidelity of recall will be discussed in Section IV-C.

A *write operation* begins with the recall of the word-line significance vector as in (13). Then, the write operation into the data memory D is

$$D := D \cup \theta(wy^T, t_D) \tag{15}$$

where $\cup$ indicates logical OR. We term this the *thresholded outer product* learning rule. The threshold t_D is defined in (10) in order to create a trapezoidal area within D. The use of logical OR causes the occupancy of the data memory to increase monotonically, which means that after a certain number of write operations most of the weights will be set to 1, resulting in poor recall performance. However, we will show that careful settings for the parameters can maintain good performance for several

TABLE I
NUMBER OF CODES REPRESENTABLE UNDER BINARY AND N-OF-M SCHEMES

	M-bit binary	Unordered N-of-M	Ordered N-of-M
Number of codes	2^M	C_N^M	$M!/(M\text{-}N)!$
11-of-256	10^{77}	10^{19}	10^{26}
200-of-1000	10^{301}	10^{216}	10^{591}

thousand write operations. This will be apparent in the results presented in Section IV-C.

IV. ANALYSIS OF THE RANK-ORDER CODED SDM

A. Information Content of Rank-Order Codes

If rank-order codes are to be stored in an associative memory, we require that they are recovered with little error; we will use the dot product and threshold as the basis for comparisons as discussed earlier. When a threshold less than 1 is used, given any particular reference code, a number of rank-order codes will have a dot product with that reference code that exceeds the threshold. The information conveyed by a matching code will be reduced accordingly. In this section, we analyze the information content of rank-order codes under the dot-product threshold matching criterion.

A valid N-of-M binary code consists of exactly N asserted bits selected from a total of M bits. N-of-M encoding combines a relatively high information capacity with intrinsic self-timing properties [16]. Without external synchronization it is impossible to detect when a binary code has stopped changing state and is valid, whereas an N-of-M code is valid when exactly N of the M bits have been asserted or, in the case of a neural layer, when exactly N of the M neurons have fired. Furthermore, for small values of N, sparse codes provide relatively high representational capacity with only a few active bits.

Rank-order codes share the same properties as unordered N-of-M codes—intrinsic self-timing, and high information capacity with a few active bits—with the addition that the information capacity is greatly increased; there are $N!$ times as many rank-order N-of-M codes as there are unordered N-of-M codes.

Table I shows formulas for the numbers of codes for M-bit binary, and unordered and ordered N-of-M schemes, together with the values these formulas yield for the examples of 11-of-256 codes (used in the rest of this paper) and 200-of-1000 codes. Taking the log of these values tells us the information content of the coding scheme. The information content of a rank-ordered N-of-M code is $\log_2(M!/(M-N)!)$ bits, which is $\log_2(N!)$ greater than the information content of an unordered code. For example, an 11-of-256 rank-order code carries 87.7 bits of information whereas the unordered 11-of-256 code carries 62.4 bits. The 200-of-1000 case illustrates that a rank-order code may have more information carrying capacity than a binary code of the same length.

Let us consider all possible rank order N-of-M codes. When compared with a selected reference code each would have a definite scalar product. We can find the information content I (measured in bits) associated with a particular threshold t by counting the number of codes $c(t)$ whose scalar product with the reference code is greater than or equal to the chosen threshold

$$I(t) = \log_2\left(\frac{M!}{(M-N)!}\right) - \log_2(c(t)). \tag{16}$$

If the threshold $t = 1$, then the number of codes $c(t) = 1$ and the information content $I(t) = \log_2(M!/(M-N)!)$. Thus, the information content can be increased by increasing M while holding N constant, so keeping the activity and, in a physical system, the power requirement constant.

In calculating the scalar product between the reference code and the test code only the overlapping nonzero values need be considered. We can divide the whole range of test codes into groups determined by the number (R) of overlapping values ranging from N down to zero. Within each group we can choose the R values from each code which overlap; there are C_R^N different ways of choosing each set and these may be permuted to yield $R!$ different arrangements each having its own scalar product. Thus, there are $N!^2/R!(N-R)!^2$ scalar products to calculate. All we need now is to calculate the number of test codes which give rise to each scalar product and this is determined by the number of possible arrangements of the other (zero) terms. There are $(M-N)!$ ways of arranging the 0's in the reference code. In the test code, the number of ways we can arrange the $N-R$ unused items among the $M-N-N+R$ 0's is $(M-N)!/(M-2N+R)!$. Thus, each scalar product is repeated $(M-N)!^2/(M-2N+R)!$ times.

Fig. 4 shows how the information content of an 11-of-256 rank-order code with a geometric significance ratio of 0.9 varies with the (normalized) dot product. The equivalent data is also shown for an unordered 11-of-256 code for comparison—here, the dependency of information content on dot product is stepped due to the discrete nature of the Hamming distance metric.

B. Address Decoder Ambiguities

When multiple address decoders sample exactly the same subset of the active address inputs they produce identical activations, thereby creating an ambiguity in the ensuing rank order. The number of clashes in the address decoder activation levels is illustrated in Fig. 5, where it can be seen that the problem shows a decreasing trend as the number of 1's in each address decoder is increased, although the trend is by no means monotonic.

Fig. 5 was generated using statistical sampling, but we can understand it by observing that it is displaying the expected number of ambiguities in the top v values from a random sample of W values taken from all possible address decoder activations. More generally, we have an address input vector of length M where exactly N of the M inputs are nonzero and follow a geometric progression with factor σ. We select a of the M values and sum them to form the activation of one address decoder. The sums from all possible arrangements are sorted into numerical order and the top fraction $f = v/W$ are selected. Within this

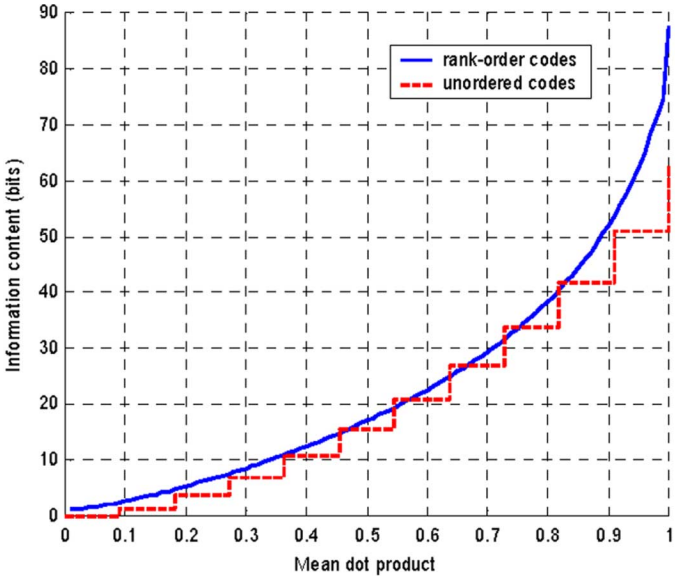


Fig. 4. Information content (in bits) of 11-of-256 rank-order (solid line) and unordered (dashed line) codes versus the normalized mean dot product using a geometric significance ratio of 0.9 for the rank-order codes. A dot product of 1 corresponds to an exact match with an information content of 87.7 bits for the rank-order code, 62.4 bits for the unordered code; a threshold below 1 treats codes that are close (as measured by their separation in the 256-dimensional significance vector space) as conveying the same information, thereby reducing the total number of distinguishable codes and, hence, the information content of each code.

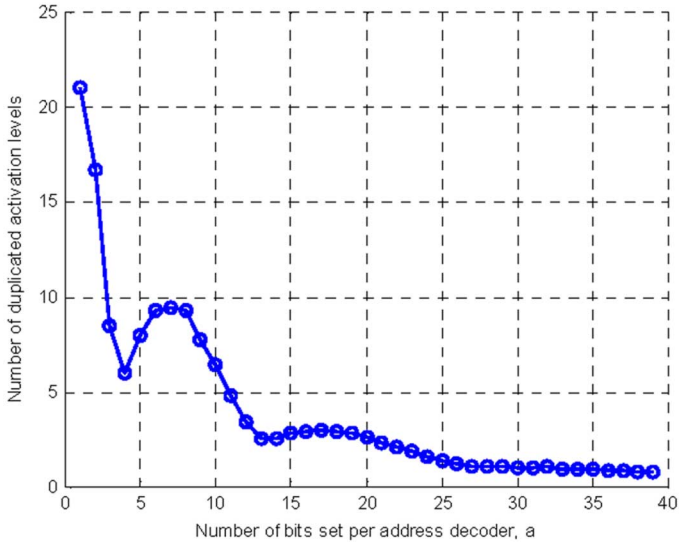


Fig. 5. Average number of duplicated address decoder activation levels among the top $v = 23$ address decoders as a function of the number a of 1's in each address decoder. The inputs are rank-order 11-of-256 codes with a geometric significance ratio of 0.9, the address decoders use a-of-256 binary weights, and there are $W = 4096$ address decoders.

group, a random selection of v is made. The number of duplicates is the difference between the total number selected v and the number of distinct values represented in the selection.

We calculate all the possible sums, noting that each is a sum of M components (many of which are zero). Since the components are sampled from a geometric series, each sum can be formed only one way. If $a \geq N$, the maximum number of distinct sums is 2^N.

Next, we select the fg cases (where g is the total number of cases) with the highest sums by first sorting them in order of value and then counting off the number of ways each sum could be generated until the total comes to fg. This group has a number of distinct sums, say U_i for $i = 1$ to V, each with a number of ways ω_i that the sum can be generated.

Now, let us consider the problem of counting duplicates when sampling from a population. If the probability of a particular object being chosen is r, then, the probability of it being chosen q times in s samples is given by the binomial distribution and the expected number of samples for any particular object is $\bar{q} = rs$. If the object is chosen q times, the number of duplicates δ is $q-1$ unless $q = 0$, in which case $\delta = 0$. Therefore, the expected number of duplicates is

$$\bar{\delta} = \bar{q} - 1 + (1 - r)^s. \tag{17}$$

From this, we can deduce the total number of duplicates in our address decoder activation levels

$$\delta_{\text{tot}} = v - V + \sum_{i=1}^{V} \left(1 - \frac{\omega_i}{fg}\right)^v. \tag{18}$$

This formula has been found to give the same results as the numerical sampling method used to produce Fig. 5. In order to reduce sorting ambiguities to a level that has minimal impact on the memory performance we employ address decoders with $a = 21$ bits set; this value is not intended to be optimal.

The characteristic "humps" displayed in Fig. 5 arise from the discrete nature of the calculation. For any given number of bits set in the address decoder, only a fraction of these will match the input word. The greatest contribution to the number of duplicates will tend to come from the set where, say, exactly k matches occur. A larger number of bits in the address decoder will give rise to a case where the number of duplicates is dominated by $k + 1$ matches. Between these two cases both sets with k matches and sets with $k + 1$ matches contribute significantly. Thus, as the number of address decoder bits increases the number of duplicates alternates between arising from a single set and arising from a pair of adjacent sets, giving rise to the oscillatory nature of the curve.

C. SDM Performance

The performance of a rank-order SDM under error-free input conditions is shown in Fig. 6, along with the performance of an equivalent memory optimized to use unordered codes [15]. The graph illustrates how the occupancy (the proportion of bits in the data memory set to 1) increases as the number of stored symbols increases. As the occupancy increases the quality of the recovered data decreases, where quality is indicated by the mean scalar product of the significance vectors of the data readout with those of the data originally written. This quality can be converted into a measure of the information per symbol (referring back to Fig. 4) which, when multiplied by the number of stored symbols, yields the usable information content of the memory. This can in turn be divided by the number of bits in the data memory to give an information efficiency measure

$$\eta(z) = \frac{I(Q(z)) \cdot z}{W \cdot M} \text{ bits/bit} \tag{19}$$

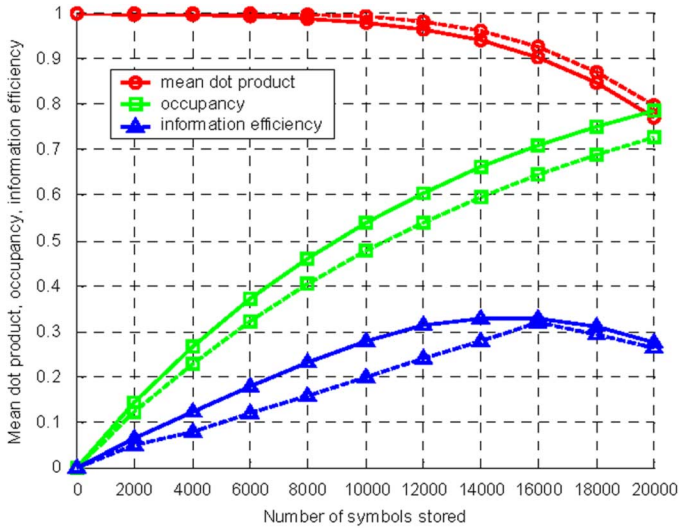


Fig. 6. Performance of binary rank-order (solid lines) and unordered (dashed lines) SDMs. The address and data symbols employ 11-of-256 codes and both memories have 10 000 hard locations. In the rank order memory, the address decoders employ 21-of-256 codes, 23 word lines are active in each read and write operation, and a geometric significance ratio of 0.9 is used throughout. The unordered memory uses address decoders with 24-of-256 codes and a match threshold of 5 bits, resulting in an average of 15 active word lines. The information efficiency is the number of bits of information stored per writable bit of data memory; the mean dot product is a measure of the fidelity of the output from the memory; the occupancy is the proportion of writable bits in the data memory that have been set to 1.

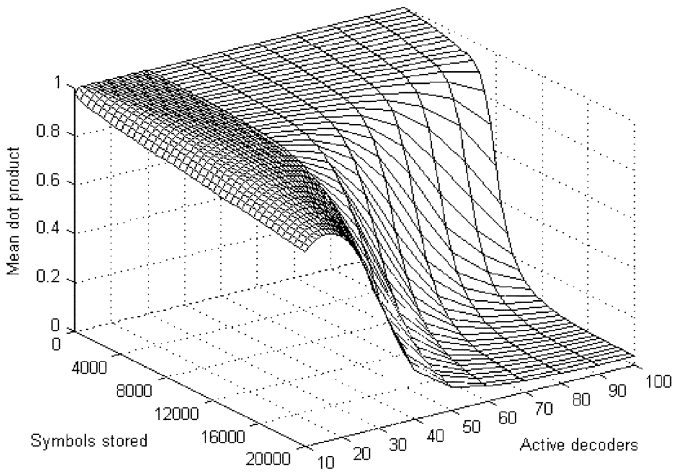


Fig. 7. Output quality (mean dot product) of a binary rank-order SDM with 10 000 address decoders as a function of the number of stored symbols with 10–100 address decoders active in each read and write operation. With ten active address decoders there is insufficient redundancy and the output quality is low. With 40 or more active address decoders the memory fills up and interference between stored symbols causes a loss of quality. The optimum number of active decoders is around 20, reducing somewhat as the number of stored symbols increases.

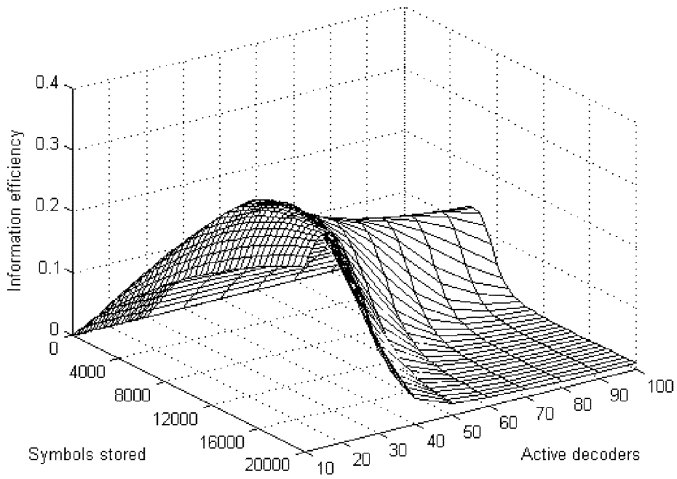


Fig. 8. Information efficiency of a binary rank-ordered SDM with $W = 10\,000$ address decoders as a function of the number of symbols stored with $v = 10$–100 address decoders active in each read and write operation. With ten active decoders the efficiency continues to rise, but at the expense of output quality (see Fig. 7). The optimum number of active decoders is again in the region of 20.

where z is the number of stored symbols, $Q(z)$ is the quality of the output as defined previously, I is the information content of each symbol as a function of the output quality as given by (16), W is the number of rows, and M is the number of columns in the data memory.

The information efficiency displays a maximum value of around 0.33 bits/bit, somewhat higher than the maximum information efficiency of the unordered N-of-M SDM with the same memory size. The maximum in the information efficiency is a consequence of the quality of the recovered data falling faster than the number of stored symbols increases, beyond a certain memory loading.

The information content of a binary correlation matrix memory is at a maximum when the occupancy is 50% [4], but the information efficiency maximum here is at an occupancy of around 65%. This suggests that the memory modeled here is not optimized with respect to all of its parameters, but we have not yet been able to improve on this performance.

V. Scalability and Optimization

The performance of the rank-order SDM depends on several parameters, and in this section, we explore the parameter space and investigate the way the performance of the memory scales with its size.

In Fig. 7, we see the effect of varying v the number of active address decoders. With $v = 10$ active decoders the output quality is very low as there is insufficient redundancy in the stored data for it to be recovered reliably. With 40 or more active decoders the memory is filled too rapidly and interference between stored symbols adversely affects performance. The optimum number of active decoders is around 20. (This result depends only very weakly on the total number of address decoders.)

The impact of the number of active address decoders on the information efficiency of the memory is illustrated in Fig. 8. Again, we see the optimum performance is achieved with around 20 active decoders, although the efficiency of the memory with ten active decoders is continuing to improve at the expense of a very low-quality output (as seen in Fig. 7). In fact, the peak information efficiency occurs in a memory with $v = 16$ active address decoders, but this results in poor output quality, and $v = 23$ active address decoders (as used to produce the results shown in Fig. 6) gives a good compromise between information efficiency and output quality.

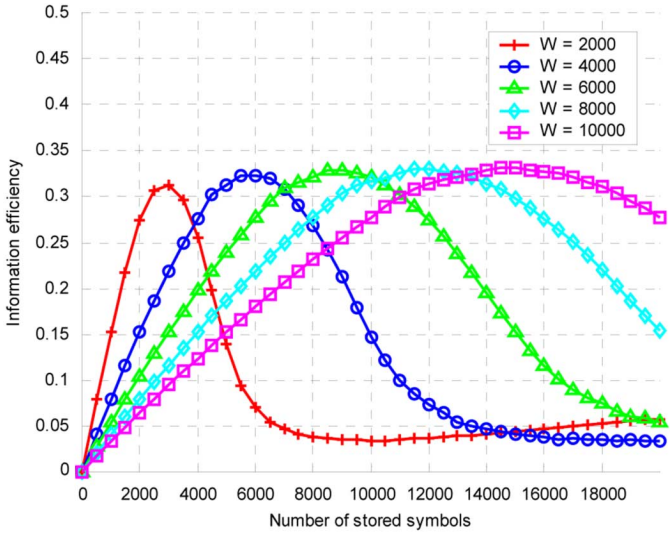


Fig. 9. Information efficiency of a binary rank-ordered SDM with $v = 23$ active address decoders as a function of the number of stored symbols where the memory has a total of $W = 2000$ to 10 000 address decoders. The performance is almost independent of the size of the memory, improving slightly with the larger sizes. This demonstrates the scalability of the SDM.

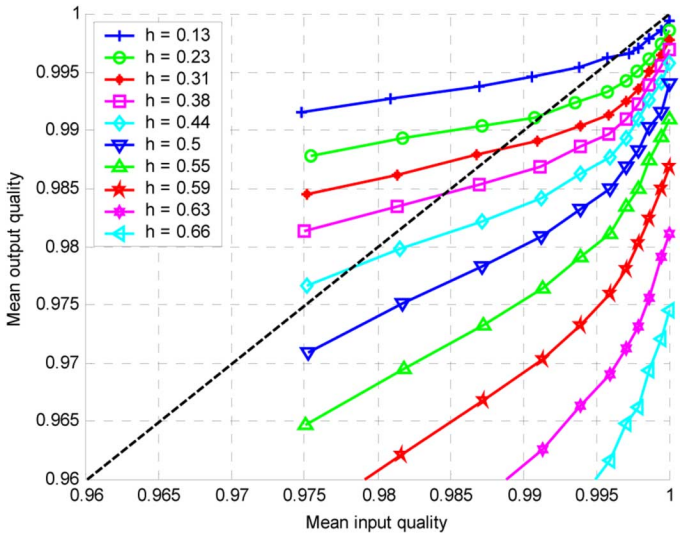


Fig. 10. Quality of the output (measured as the mean dot product of the output with the correct output) of a rank-order SDM versus the quality of the input address set. The memory has 4096 hard locations and $v = 50$ word lines are active in each read and write operation. Address and data symbols employ 11-of-256 codes. A geometric significance ratio of 0.9 is used throughout the memory except for the word lines which use a geometric significance ratio of 0.9655 in order to increase the order-preserving property of the memory in the presence of noisy inputs. The diagonal line indicates equal input and output quality; the operating region above and to the left of this line (where the memory has lower occupancy) is where the output from the memory is, on average, of higher quality (i.e., contains fewer errors) than the input.

The scalability of the memory is demonstrated in Fig. 9, which shows how the information efficiency of an SDM with $v = 23$ active address decoders varies with the number of stored symbols for several different memory sizes, from $W = 2000$ to $W = 10\,000$ address decoders (and data storage locations) in total. The peak information efficiency is almost independent of the size of the memory, increasing slightly with the larger memory sizes. The number of stored symbols at the peak efficiency increases proportionately with the size of the memory.

VI. Input Errors

So far we have considered the operation of the rank-order SDM only under error-free input conditions. If, for example, the address used to recover information from the memory is not identical to the address originally used to store the information, but has been corrupted by some source of noise or error, then how will the memory perform?

In order to assess the performance of the memory under error conditions we need to define the sorts of errors that we might expect to arise. The approach we have taken is to test the memory with a set of addresses whose quality is defined by the mean scalar product of a (reference) set used to write the data with the (noisy) set used to recover the data. We can then assess whether, and under what conditions, the quality of the recovered data is better than the quality of the set of addresses used to recover that data. In an autoassociative memory, this will indicate whether repeated cycling through the memory will result in convergent or divergent behavior.

The generation of a set of addresses with a given mean scalar product with a reference set is potentially problematic as order and/or population errors can be used in any mix to reduce the mean scalar product below 1. We approach this by employing an autoassociative SDM where, as we increase its loading, we read out sets of data (where the stored values were originally the same as the addresses) with progressively decreasing quality. We then use these sets of data as the address sets for testing the performance of a second SDM.

The results of this investigation are shown in Fig. 10. Here, the input address quality is plotted against the output data quality for a range of memory occupancies (corresponding to the number of stored patterns increasing from 400 at $h = 0.13$ to 4000 at $h = 0.66$ in steps of 400). The results show that the memory has convergent properties when the occupancy is below 50%, with stable quality level (indicated by the intersection of the curve with the broken diagonal line) decreasing as the occupancy increases towards this level. For occupancies above 50%, the memory is divergent and the quality of the output is always lower than the quality of the input under the operating conditions investigated here. While some of the curves at higher occupancies have shapes that suggest they may intercept the diagonal in Fig. 10, this will be at input and output qualities that are too low for the rank ordering to offer any benefit over unordered codes, so this has not been investigated further. The memory used to produce these results has a larger number of active word lines (50 rather than 23) in order to provide a greater redundancy of storage and, hence, of error tolerance than the memory optimized for error-free addresses, and uses a higher significance ratio for the word lines in the data memory to cause the difference between the number of bits set for any data bit and the number set for the adjacent bits—the skew s—to be 3 rather than 1, in order to offer greater resilience to noise in the storing of the order information.

These results offer a different perspective on the useful information capacity of the memory. If the memory is viewed as a component in a multilayer system operating on noisy data,

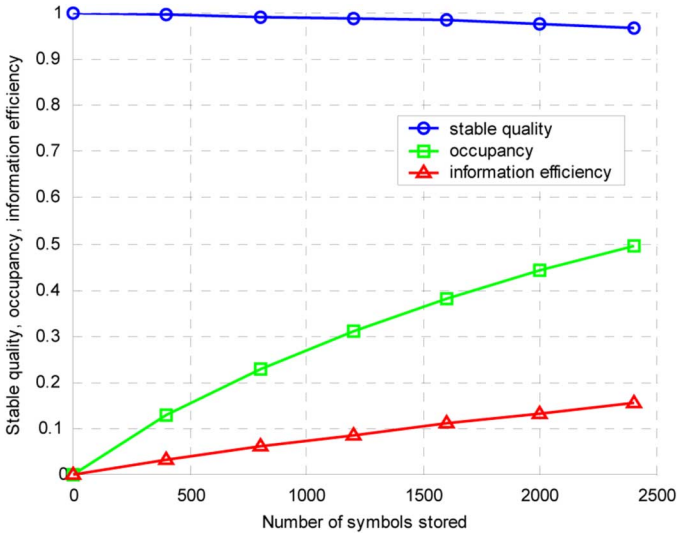


Fig. 11. Performance of the rank-order SDM with input and output errors. The memory has 4096 hard locations and $v = 50$ word lines are active in each read and write operation. Address and data symbols employ 11-of-256 codes; the address decoders employ a 21-of-256 code. A geometric significance ratio of 0.9 is used throughout the memory except for the word lines which use a geometric significance ratio of 0.9655. The operating point of the memory for these tests is where the input address and output data qualities are equal. The peak information efficiency is 0.15 bits/bit when 2400 symbols are stored in the memory; at this operating point the mean input and output quality is 0.967 and each symbol stores 67.6 bits of information.

then it is important that it does not degrade the quality of the information that passes through it. Thus, we can view the stable quality point as indicative of the maximum usable information content of the memory. Using this view, we obtain the results shown in Fig. 11. The peak information efficiency is lower than for the memory with error-free addressing, but 0.15 bits/bit is still achieved.

VII. Implementation With Spiking Neurons

A. Spiking Rank-Order Neural Layer

A rank-order neural layer should display sensitivity to the rank order of the address input and it should produce a rank-order coded output. Sensitivity to the input address rank-order is realized by shunt inhibition that causes the gain of every neuron to diminish with each firing input [17] (see Fig. 12). In our SDM model presented in this paper, this process is abstracted into the significance vector representation of the input address. An individual neuron may be tuned to a particular input order by assigning a monotonic decreasing series of synaptic weights with the largest connecting the first input neuron to fire, the second largest the second to fire, and so on. However, as discussed in Section III-C, a population of neurons can be sensitive to input rank-order using simple binary weights.

The response of each neuron is the scalar (dot) product of the input significance vector and the individual neuron's synaptic weight vector. The output rank order can then be generated by ordering the individual responses with a cutoff after the required number. The output from the neuron most closely tuned to the input will fire first, the second most closely tuned will fire

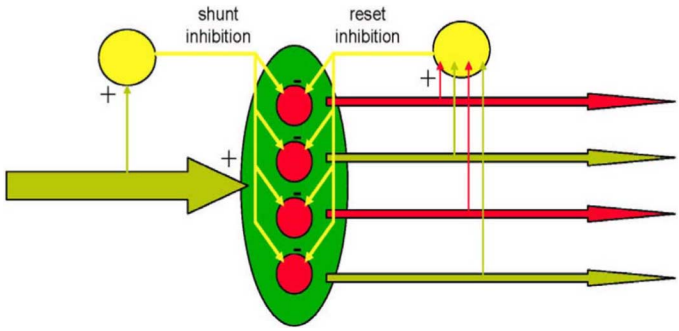


Fig. 12. Possible architecture capable of detecting and generating rank-ordered N-of-M codes. The feedforward shunt inhibition on the input side is used to reduce the gain of the main bank of neurons, thereby making them sensitive to the rank order of the inputs. The feedback reset inhibition on the output side is used to prevent the number of neurons that fire in each burst exceeding N.

second, and so on until the required number of neurons has fired. The cutoff can be implemented using inhibitory feedback neurons that count the firing outputs and then inhibit further firing once the required number of outputs has fired [17] (again illustrated in Fig. 12), but, here, we again abstract the details into the rank-order model by enforcing a cutoff. This is a form of generalized *winner-take-all* circuit that has been studied in the past [13]. We have shown elsewhere that the *abstract* rank-order model can be implemented using systems of spiking neurons with appropriately chosen dynamics [18].

Biological neurons are noisy and will not generally conform strictly to the regimented rank-ordered firing scheme that is the basis of the work described here. Erroneous spikes are likely to upset our model significantly, so we cannot claim at this stage that the model described here offers an accurate analogy of a biological memory system. Similarly, real data is unlikely to perform as well as the random data we have used as it will not be distributed evenly in the high-dimensional space the data occupies, and data where the number of nonzero elements is not fixed will create problems for the dot-product metric we have used. Further work is required to investigate the possibility that the rank-order model might offer a partial description of a biologically plausible system.

B. Geometric Significance Ratio

The use of a geometric significance ratio, as described earlier, can be supported (as a reasonable approximation) on the following grounds. First, we require that the shunt inhibition (see Fig. 12) has the effect of dividing the total excitory input by some factor [19], which we will implement using multiplication by a factor less than 1. This means that the activation Ω of an individual neuron (here, we use the leaky integrate-and-fire model) can be described in the following form:

$$\Omega = -\Omega/\tau_\Omega + \left(\sum w_{ij}x_j\right) G \tag{20}$$

where τ_Ω is the "leaky" time constant, w_{ij} are the input connection weights, x_j are the inputs (assumed here each to be a time series of delta function spikes), and G is the shunting inhibition factor.

We assume further that the inhibition follows a similar leaky integrating form

$$\dot{G} = (1 - G)/\tau - \alpha \sum x_j \tag{21}$$

where the resting value (in the absence of inputs) is 1, τ is the time constant, and $\alpha < 1$ is a scale factor.

If the input burst representing a rank-order code is equally-spaced over time, with a separation t between consecutive input spikes, then, the nth input will be modulated by

$$G_n = 1 - k \cdot e^{-t/\tau} \cdot \left(1 + e^{-t/\tau} + \cdots + e^{-(n-2)t/\tau}\right) \tag{22}$$

where, if we use $\sigma = e^{-t/\tau}$ and, then, set $\alpha = (1/\sigma) - 1$, we obtain the geometric relationship $G_n = \sigma^{n-1}$, as required to support the geometric significance ratio model. Now, clearly it has required several assumptions to get to this result, but this does show that the geometric significance ratio may be a reasonable working approximation to a plausible implementation using leaky integrate-and-fire neurons.

VIII. Conclusion

We have presented an SDM that is capable of storing and recovering rank-order codes with a high degree of robustness. The memory employs only binary synaptic weights, those in the first (address decoder) layer being fixed and those in the second (data memory) layer being updated to store data as it is presented to the memory. For the toy learning problems we used, such a memory can store data with an information efficiency of over 0.33 bits/bit of data memory; it is expected that when the memory is scaled up to inputs of several thousand bits, efficiency will be higher still, though perhaps not as high as the asymptotic efficiency of an unordered correlation matrix memory [12].

We have adopted a measure of similarity between two rank-order codes based on the dot product of their representations as significance vectors, a measure that captures errors in both order and population, and defined the quality of a set of codes in terms of the mean of the dot product of each member of the set with the corresponding member of an error-free reference set. Using this measure, we have shown that under low load a memory can produce output data that has higher quality than the input data, i.e., it has error-correcting abilities in the presence of noise. The information efficiency of the memory at the stable quality point (where the output data quality is no worse than the input address quality) can be over 0.15 bits/bit of data memory.

We have thus shown that a simple binary SDM can be used to store and recover rank-ordered codes with good efficiency and that the memory has appreciable error-recovery capabilities that render it usable in complex noisy systems.

References

[1] P. Kanerva, *Sparse Distributed Memory*. Cambridge, MA: MIT Press, 1988.

[2] L. A. Jaeckel, "A class of designs for a sparse distributed memory" NASA Ames Research Centre, RIACS Tech. Rep. 89.30, 1989.

[3] T. A. Hely, D. J. Willshaw, and G. M. Hayes, "A new approach to Kanerva's sparse distributed memory," *IEEE Trans. Neural Netw.*, vol. 8, no. 3, pp. 791–794, May 1997.

[4] D. J. Willshaw, O. P. Buneman, and H. C. Longuet-Higgins, "Non-holographic associative memory," *Nature*, vol. 222, pp. 960–962, 1969.

[5] T. Kohonen, "Correlation matrix memories," *IEEE Trans. Comput.*, vol. C-21, pp. 353–359, 1972.

[6] P. Suppes, "Representations and models in psychology," *Annu. Rev. Psychol.*, vol. 45, pp. 517–544, 1994.

[7] M. Rehn and F. Sommer, "Storing and restoring visual input with collaborative rank coding and associative memory," *Neurocomput.*, vol. 69, pp. 1219–1223, 2006.

[8] M. Lengyel, J. Kwag, O. Paulsen, and P. Dayan, "Matching storage and recall: Hippocampal spike timing-dependent plasticity and phase response curves," *Nature Neurosci.*, vol. 8, no. 12, pp. 1677–1683, 2005.

[9] S. Thorpe, A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," *Neural Netw.*, vol. 14, pp. 715–725, 2001.

[10] L. Perrinet, M. Samuelides, and S. Thorpe, "Coding static natural images using spiking event times: Do neurons cooperate?," *IEEE Trans. Neural Netw.*, vol. 15, no. 5, pp. 1164–1175, Sep. 2004.

[11] W. Maass, "Computing with spiking neurons," in *Pulsed Neural Networks*, Maass and Bishop, Eds. Cambridge, MA: MIT Press, 1998, ch. 2, pp. 55–85.

[12] G. Palm and F. T. Sommer, "Associative data storage and retrieval in neural networks," in *Models of Neural Networks III: Association, Generalization, and Representation*, E. Domany, J. L. van Hemmen, and K. Schulten, Eds. Berlin, Germany: Springer-Verlag, 1995, pp. 79–118.

[13] W. Maass, "On the computational power of winner-take-all," *Neural Comput.*, vol. 12, pp. 2519–2535, 2000.

[14] J. Austin and T. J. Stonham, "An associative memory for use in image recognition and occlusion analysis," *Image Vis. Comput.*, vol. 5, no. 4, pp. 251–261, 1987.

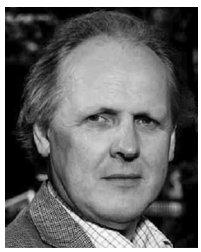
[15] S. B. Furber, W. J. Bainbridge, J. M. Cumpstey, and S. Temple, "A sparse distributed memory based upon N-of-M codes," *Neural Netw.*, vol. 17, no. 10, pp. 1437–1451, 2004.

[16] T. Verhoeff, "Delay-insensitive codes—An overview," *Distributed Comput.*, vol. 3, pp. 1–8, 1998.

[17] S. J. Thorpe, R. Guyonneau, N. Guilbaud, J. M. Allegraud, and R. Vanrullen, "SpikeNet: Real-time visual processing with one spike per neuron," *Neurocomput.*, vol. 58–60, pp. 857–64, 2004.

[18] J. Bose, S. B. Furber, and J. L. Shapiro, "A system for transmitting a coherent burst of activity through a network of spiking neurons," in *Proc. XVI Italian Workshop Neural Netw. (WIRN)*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2006, vol. 3931, pp. 44–48.

[19] F. S. Chance and L. F. Abbott, "Divisive inhibition in recurrent networks," *Network: Comput. Neural Syst.*, vol. 11, pp. 119–129, 2000.



Stephen B. Furber (M'99–SM'01–F'05) received the B.A. degree in mathematics and the Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K., in 1974 and 1980, respectively.

He is the ICL Professor of Computer Engineering in the School of Computer Science, the University of Manchester, Manchester, U.K. From 1981 to 1990, he worked in the hardware development group within the R&D Department at Acorn Computers Ltd., and was a Principal Designer of the BBC Microcomputer and the ARM 32-bit RISC microprocessor, both of which earned Acorn Computers a Queen's Award for Technology. Upon starting his work at the University of Manchester in 1990, he established the Amulet research group with interests in asynchronous logic design and power-efficient computing, which merged with the Parallel Architectures and Languages group in 2000 to form the Advanced Processor Technologies (APT) group. The APT group is supported by an Engineering and Physical Sciences Research Council (EPSRC) Portfolio Partnership Award.

Prof. Furber is a Fellow of the Royal Society, the Royal Academy of Engineering, the British Computer Society, and the Institution of Engineering and Technology. He is a Chartered Engineer.

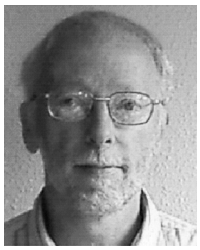
Gavin Brown received the Ph.D. degree from the University of Birmingham, Birmingham, U.K., for research on neural network ensembles, in 2004.

He holds a Career Development Fellowship in Cognitive Systems at the University of Manchester, Manchester, U.K. His current research interests include classifier combination techniques, speciation in evolutionary algorithms, probabilistic models, and applications of machine learning to computer architecture and systems problems.

Dr. Brown received the British Computer Society Distinguished Dissertation Award in 2004.

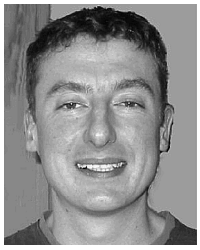
Joy Bose received the B.Eng. degree in computer science from Motilal Nehru Regional Engineering College (now MNNIT), Allahabad, India, in 2002. He is currently working towards the Ph.D. degree in computer science at the University of Manchester, Manchester, U.K.

His research involves building an online predictive sequence machine out of spiking neurons.



John Michael Cumpstey received the B.A. degree in physics from the University of Oxford, Oxford, U.K., in 1968.

After working in the nuclear power industry for 26 years, he joined the University of Manchester, Manchester, U.K., in 1994, where he is currently a Research Fellow in the School of Computer Science. His current research interests include the mathematical modeling of artificial spiking neural networks.



Peter Marshall received the First Class M.Eng. degree after studying microelectronics systems engineering at the Departments of Computation and Electrical Engineering and Electronics, the University of Manchester Institute of Science and Technology (UMIST), Manchester, U.K., in 1994.

He then became a Consultant to the consumer electronics industry, with clients including Pace Micro Technology. In 2002, he became a Research Fellow in the Department of Computer Science, the University of Manchester, Manchester, U.K., specializing in neuron simulation with a focus on SDM. He then became Software Manager for Fusion Digital Technology, a startup venture specializing in digital television, and currently, he is a Co-Director of Applied Software Engineering, a growing independent consultancy business specializing in embedded systems.

Jonathan L. Shapiro received the Ph.D. degree in physics from the University of California at Los Angeles (UCLA), Los Angeles, in 1986.

He is a Senior Lecturer in Computer Science at the University of Manchester, Manchester, U.K., where he heads the Artificial Intelligence Research Group. His research is in mechanisms of learning and adaptation and probabilistic modeling, with applications to both artificial and natural systems. Applications include navigation, exploration, and communication in robots, evolutionary dynamics in population genetics and in optimization algorithms, modeling of time perception and reinforcement learning in animals, and optimization by probabilistic learning systems.