Sparse Distributed Memory using *N*-of-*M* Codes

Steve B. Furber, W. John Bainbridge, J. Mike Cumpstey and Steve Temple

Department of Computer Science,
The University of Manchester,
Oxford Road,
Manchester M13 9PL UK

Corresponding author:

Professor S B Furber

Department of Computer Science,

The University of Manchester

Oxford Road

Manchester M13 9PL, UK

tel: (+44) 161 275 6129

fax: (+44) 161 275 6236

email: sfurber@cs.man.ac.uk

Running title. *N*-of-*M* Sparse Distributed Memory

Abstract

An analysis is presented of a sparse distributed memory (SDM) inspired by that described by Kanerva (1988) but modified to facilitate an implementation based on spiking neurons. The memory presented here employs sparse binary *N*-of-*M* codes, unipolar binary synaptic weights and a simple Hebbian learning rule. It is a two-layer network, the first (fixed) layer being similar to the 'address decoder' in Jaeckel's (1989) 'hyperplane' variant of Kanerva's SDM and the second (writeable) 'data store' layer being a correlation matrix memory as first proposed by Willshaw et al (1969). The resulting network is shown to have good storage efficiency and is scalable. The analysis is supported by numerical simulations and gives results that enable the configuration of the memory to be optimised for a range of noiseless and noisy environments.

Keywords: Neural networks; Spiking neurons; Sparse distributed memory; Associative memory; Correlation matrix memory; Unipolar weights; *N*-of-*M* codes.

Mathematical Symbols

A – the number of bits in the input address.

$\mathbf{A}$ – the i-of-A sparse binary input address (cue) vector.

A – the binary-valued $A \times W$ address decoder matrix (with fixed weights).

a – the number of 1s in each row of the address decoder; each address decoder is programmed with an a-of-A code mask that selects a bits from the input address.

C_b^a – the combinatorial operator = $\frac{a!}{b!(a-b)!}$.

D – the number of bits in a data word.

D – the binary-valued $W \times D$ data storage matrix (with updatable weights).

d – the number of 1s in a data word (forming a d-of-D sparse binary code).

$\mathbf{D_{in}}$ – the d-of-D sparse binary input (write) data vector.

$\mathbf{D_{out}}$ – the d-of-D sparse binary output (read) data vector.

$E(x)$ – the expected value of x.

E_c – the expected number of data words read out without error (in a sequence of reads where other words do contain errors).

f – the number of errors (1s in an incorrect position) in a d-of-D data output code.

h – occupancy; the proportion of data storage matrix bits set to 1 by a series of write operations.

I – the recoverable information content of the memory, in bits.

i – the number of 1s in an input address; each input address is an i-of-A sparse binary code.

k – a scaling factor used in the analysis of the address decoder under erroneous input conditions.

n – the number of misplaced 1s in a noisy input (read-out) address.

P_c – the probability that every data word written into the memory is read out without error.

$p(x)$ – the probability of x.

T – the address decoder threshold, applied to each row of the address decoder to determine whether or not it is active.

W – the number of address decoder rows and the number of word locations in the data memory.

W – the *w*-of-*W* sparse binary vector output by the address decoder indicating the *w* active address decoder rows.

w – the number of active rows of the address decoder; also the number of locations in the data memory that each data word is written into.

$\hat{w}$ – the expected, or mean, value of w.

w' – a particular value of w.

w_s – the number of address decoder rows that are active in a read operation (with input cue errors) that were also active in the corresponding write operation - the 'signal' component of **W**.

w_n – the number of address decoder rows that are active in a read operation (with input cue errors) that were *not* active in the corresponding write operation - the 'noise' component of **W**.

X_0 (X_1) – the activation levels, during a read operation, of a data output that was written as 0 (1) in the corresponding write operation.

X_t – the threshold activation level in a read operation that determines whether a particular data output bit should deliver a 0 or a 1.

Z – the number of data words written into the memory.

η – the storage efficiency of the memory, expressed as the ratio of the number of bits of information stored successfully (so that they can be retrieved) to the number of bits needed to realise the adjustable weights (in the array of data storage cells).

$\sigma^2(x)$ – the variance of x.

1. Introduction

Associative memories based on neural structures have been studied for many years. In this paper we investigate a form of sparse distributed memory (SDM) that has takes its inspiration from Kanerva's (1988) SDM, but is substantially modified to support an implementation based on spiking neurons. The resulting memory can be viewed as combining an address decoder layer similar to that in Jaeckel's (1989) 'hyperplane' variant of Kanerva's SDM with a data store layer that is a binary correlation matrix memory as first described by Willshaw et al (1969).

Conventional (dense) binary codes as used in Kanerva's SDM are ill-suited to spiking neural implementations. If a spike represents a 1 then a neuron can readily detect the 1s in the input code, but it has no simple way to detect the 0s. This problem is avoided if the code has a fixed number of 1s, since once the 1s have been detected the 0s are located implicitly. This leads us to the use of *N*-of-*M* codes, where in a population of *M* neurons exactly *N* fire for each symbol. The suitability of *N*-of-*M* codes for spiking neural implementations is closely related to their self-timing properties (Verhoeff, 1998) – the completeness of the data is implicit in the coding – so consecutive layers of neurons 'know' when they have complete input information and can use this to trigger the generation of their outputs.

N-of-*M* codes have been studied extensively in the context of neural network coding, both in their precise form where *N* is fixed, generally as a result of an *N*-max algorithm, and in their approximate form where *N* is subject to statistical variation, for example as a result of each neuron firing independently with probability *N*/*M*. In most cases the coding is sparse ($N << M$) and this results in a constant low average neural activity.

We employ *N*-of-*M* codes throughout the SDM presented in this paper to yield a memory that retains the principal feature of Kanerva's memory – an address decoder layer that casts the input

symbols into a high-dimensional space in order to enhance their separability – whilst enabling a straightforward implementation based on spiking neurons. The memory is characterized both analytically and numerically to reveal its capacity and tolerance to noisy inputs, and these results indicate how the performance and noise-immunity may be optimized under various operating conditions.

1.1. *N*-of-*M* memories

Correlation Matrix Memories (CMMs) have been subject to extensive study (Willshaw et al, 1969; Schwenker et al, 1996; Casasent & Telfer, 1992; Kohonen, 1972; Lomas, 1996; Turner & Austin, 1997), and we observe that the data store layer in our memory is just a CMM with *N*-of-*M* input and output encoding. The distinctive feature of the work described here is the preprocessing of the inputs into a high-dimensional *N*-of-*M* space using an 'address decoder' layer, giving the memory a capacity that can scale independently of *M*.

N-of-*M* coding has been considered previously for various forms of associative processing. Casasent and Telfer (1992) looked at several configurations of associative processor based on a single matrix with various training algorithms and binary or analogue, unipolar or bipolar weights. They conclude, amongst other results, that *N*-of-*M* codes (which they call *L-max* encoding) are advantageous. Nadal and Toulouse (1990) studied the storage capacity of single matrix *N*-of-*M* associative memories. Davidson (1999) also considered single-matrix memories using *N*-of-*M* codes, presenting theoretical analyses of memory capacity and redundancy, and backing these up with numerical simulations. In this paper we apply similar techniques to analyse a memory comprising two matrices, considering only unipolar binary weights, as outlined in Section 4.

Aleksander and Stonham's (1979) *N*-tuple memories employ a binary-encoded addressing mechanism. The address itself is a dense *M*-bit binary code from which an *N*-bit random sample of

bits is presented to a conventional RAM, where those bits are fully decoded. The outputs from several such RAMs (each using a different random sample of address bits) are summed, the output with the highest score determining the classification of the input address. The system can also be viewed as a single binary correlation matrix memory whose row inputs are generated by an address decoder that produces an output comprising a number of 1-of-2^N codes. This approach has a lot in common with the one presented here, but it targets a different implementation technology – conventional RAMs instead of spiking neurons – and therefore ends up with a different address decoder structure.

The 'hyperplane' variant of Kanerva's memory (Jaeckel, 1989) also employs *N*-of-*M* codes, and is closely related to the memory we analyse here. It is discussed further at the end of Section 3.

1.2. Outline of paper

Section 2 presents the relevant properties of *N*-of-*M* codes. Section 3 presents a summary of Kanerva's original sparse distributed memory (which employs binary codes) and the 'hyperplane' variant that makes some use of *N*-of-*M* codes. Section 4 describes the structure and presents the theoretical analysis of the *N*-of-*M* variant of Kanerva's memory under error-free conditions, ending with a comparison of the theoretical results with numerical simulations. Section 5 extends the results to include the effects of input and data errors, and Section 6 covers the efficiency of the proposed memory. Section 7 introduces an implementation of the memory based on spiking neurons. Section 8 summarizes the paper and draws some conclusions, including a discussion of the relevance of the model to the understanding of biological systems.

2. N-of-M code properties

When comparing two random *N*-of-*M* codes, one N_1-of-*M* and the other N_2-of-*M*, the statistics of the likely match are of interest. The probability that two such random *N*-of-*M* codes share *x* 1s follows the hypergeometric distribution (Feller, 1950):

$$p(x) = C_x^{N_1} \cdot C_{N_2 - x}^{M - N_1} / \quad C_{N_2}^{M} \tag{1}$$

where C_b^a is the combinatorial operator $\frac{a!}{b!(a-b)!}$.

The number of matching 1s, *x*, will be between 0 and min(N_1, N_2), and the probability distribution defined by equation 1 has a mean given by:

$$E(x) = \frac{N_1 \cdot N_2}{M} \tag{2}$$

Figure 1 shows some typical distributions for cases where $N_1 = N_2 = N$. Note that there are two distinct modes of matching distribution, depending on whether $(N+1)^2$ is greater than or less than $M+2$; in the latter case the peak of the distribution is at zero. (This can be established by comparing $p(0)$ with $p(1)$ in equation 1).

A key observation that will be important in what follows is that the distributions for the sparse cases (e.g. N = 30 in figure 1) are not at all smooth. The number of matching 1s, *x*, can take only integer values, and so the probability of meeting a matching threshold can be controlled only coarsely by adjusting that threshold.

3. Kanerva's sparse distributed memory

Kanerva (1988) proposed an organization for a two-layer sparse distributed memory based upon a high-dimensional binary space. A binary address of, say, 1,000 bits is presented to a large set of fixed 'address decoders' that form the first layer of the memory. Each address decoder re-

sponds to any input within a specified Hamming distance of a particular input code. If the memory has a million address decoders, each could be assigned a random 1,000-bit code and the distance defined so that any input address activates approximately 1,000 decoders.

The second layer of the memory, the data store, is a matrix of up/down counters. When information is to be stored in the memory it is presented on the data store columns in binary form. The address decoder outputs connect to the data store rows. All of the counters on rows driven by active address decoders either increment their count if the column data input is 1 or decrement it if the data column input is 0. The counters on inactive rows are unchanged. Each data item stored in the memory is thereby distributed across the 1,000 locations activated by the input address.

When information is to be retrieved, an address 'close' to the address used to store the required information must be presented. All of the counts in each column that are connected to an active address decoder row are summed, and those column sums that exceed a threshold generate a 1 output. The key property endowed by the 1000-dimensional binary space is that a very high proportion of the space differs from any given point by over 400 bits. The 'critical distance' that defines the term 'close' used above can therefore be hundreds of (but fewer than 400) bits, decreasing as the number of stored patterns increases.

The 'hyperplane' variant of Kanerva's sparse distributed memory (Jaeckel, 1989) employs an input address with a fixed small number of 1s (for example, a 100-of-1000 code) and the address decoder rows are set to 3-of-1000 codes. An address decoder is active if its three 1s correspond to 1s in the input address. The data store comprises up/down counters as with the original Kanerva design. In our design we apply a threshold match criterion to each address decoder row rather than requiring a full overlap between the 1s in the input address and the decoder row, and for the data store layer we use a binary correlation matrix memory in place of the matrix of up/down counters.

The replacement of up/down counters with binary weights greatly simplifies the data store and has very little adverse effect on performance, as has been noted before (Schwenker et al, 1996; Palm & Sommer, 1996).

4. An *N*-of-*M* coded sparse distributed memory

In this section we present a sparse distributed memory that employs *N*-of-*M* codes. The organization of the memory is illustrated in figure 2. The memory comprises two matrix layers:

- the 'address decoder' layer is a matrix A connecting A address inputs to W outputs, with fixed binary weights initialised to a random a-of-A code in each row, and with a threshold T set to control the number of active hard location selector outputs, w. The address decoder firing pattern is then a w-of-W code though, as we shall see, it is generally impractical to avoid allowing some variation in w.
- the 'data memory' layer is a matrix D with W hard location selector inputs, initialised with zero weights. The write function simply sets each weight with an active hard location selector input and an active write data input to 1. The output function is 'd-max' – the d columns with the highest activation levels are selected and the remainder inhibited – to yield a d-of-D code.

The 'address' input to the memory, **A**, employs an i-of-A code, where the number of 1s (i) may or may not be equal to the number of 1s in each address decoder row (a). We shall see that there is a (small) advantage in allowing i to be independent from a.

The resulting memory system is an associative memory with similarities to other neural associative memories (Austin 1995). The principal differentiating feature is the use of a first layer to cast the inputs into a high-dimensional space (as with Kanerva's memory), the outputs of which are then processed through a correlation matrix memory (unlike Kanerva's memory). Other associative

memories have generally employed application-specific input layers configured as feature extractors (e.g. Austin & Stonham, 1987).

4.1. Data memory efficiency

The data memory layer is a form of correlation matrix memory that has been studied extensively in the past (Nadal & Toulouse, 1990; Willshaw et al, 1969; Schwenker et al, 1996; Casasent & Telfer, 1992; Kohonen, 1972; Lomas, 1996; Palm & Sommer, 1996; Turner & Austin, 1997). We present an analysis of this store again here as it forms the basis for what follows.

We assume that address and data inputs are uniformly distributed over their respective *i*-of-*A* and *d*-of-*D* sparse binary spaces. The data memory initially contains 0s, and we use the measure of occupancy, *h*, which represents the probability of an arbitrary bit in the data store being set to 1 at some point in the write process. After *Z* write operations the expected occupancy of the data store is given by:

$$h = 1 - \left[1 - \frac{w}{W} \cdot \frac{d}{D}\right]^{Z} \tag{3}$$

h is initially 0, and tends monotonically towards 1 as more data is stored in the memory.

Rewriting equation 3:

$$\ln(1-h) = Z \cdot \ln\left[1 - \frac{w}{W} \cdot \frac{d}{D}\right] \tag{4}$$

With sparse codes the term $w \cdot d / (W \cdot D)$ is small, so the number of stored data items (with no consideration yet of whether or not these stored items can be recovered) is closely approximated by:

$$Z = -\ln(1-h) \cdot \frac{W}{w} \cdot \frac{D}{d} \tag{5}$$

The total information stored in the memory after Z write operations may be identified by considering the number of symbols that can be represented by a d-of-D code:

$$I(Z) = Z \cdot \log_2[C_d^D] \text{ bits} \tag{6}$$

If we compare this capacity with the number of binary storage locations in the data array we can get an idea of the storage efficiency of the memory:

$$\eta(Z) = \frac{I(Z)}{D \cdot W} = \frac{Z}{D \cdot W} \cdot \log_2[C_d^D] \tag{7}$$

For large D, the combinatorial term can be approximated by:

$$\log_2[C_d^D] = -D \cdot \left[\frac{d}{D} \cdot \log_2\frac{d}{D} + \left(1 - \frac{d}{D}\right) \cdot \log_2\left(1 - \frac{d}{D}\right)\right] \tag{8}$$

(The term on the right-hand side of the equation in square brackets is the Shannon entropy for $p = d/D$.) For a sparse memory a combination of equations 5, 7 and 8 yields, after simplification:

$$\eta(Z) = -\log_2(1 - h) \cdot \frac{1}{w} \cdot \left[\ln\frac{D}{d} + 1\right] \tag{9}$$

This shows that the most efficient memory, at a given level of occupancy and ignoring errors, is the most sparse – a unary 1-of-D code – and operates with the smallest number of firing address decoders, $w = 1$. However, these choices adversely affect the robustness of the memory since errors will arise as soon as the number of patterns written exceeds the number of storage locations (W), at which point the occupancy is still very low ($h = 1/D$).

4.2. Data memory robustness

We consider the retrieval of a single data word following the writing of Z data words, where the retrieval address is presented without error. The hard address selector vector, **W**, will be the same for both the write and the corresponding read operation. If a data bit value was written as a 1, then

every activated weight will be set and the expected activation level of the data output column in D will be given by:

$$E(X_1) = w \tag{10}$$

If, on the other hand, the data value was written as a 0, the expected activation level of the column in D will be given by:

$$E(X_0) = w \cdot h \tag{11}$$

So, since $h < 1$, the memory can reasonably be expected to recover the written data. However, the activation level of the 0 output is subject to statistical variation, and with some probability all the weights activated will have been set by other data writes, in which case the activation level of an output that should be 0 will be the same as that of one that should be 1, leading to the possibility of a 'false 1' error. (In the absence of input errors there is no possibility of a 'false 0' error.) This only happens if every weight contributing to the 0 output is set, so the probability of a 0 data bit being read correctly is:

$$p(X_0 < w) = 1 - h^w \tag{12}$$

Thus the probability of a complete data value being read correctly, which requires that every 0 data bit is read correctly, is:

$$p(\text{data word correct}) = [1 - h^w]^{D-d} \tag{13}$$

and the probability that every stored data word can be read correctly, P_c, is:

$$P_c = [1 - h^w]^{Z \cdot (D-d)} \tag{14}$$

Clearly, as the occupancy *h* increases the probability of the error-free recovery of all of the written data decreases towards zero. The reliability of the memory is also a steep function of the number of active address decoder rows (*w*) as we shall see.

Rewriting equation 14 yields:

$$\ln(P_c) = Z \cdot (D - d) \cdot \ln(1 - h^w) \tag{15}$$

Again, h^w is very small (when *w* is significantly greater than 1 and *h* is not too close to 1) so the first term of the Taylor series gives a good approximation:

$$\ln(P_c) = -Z \cdot (D - d) \cdot h^w \tag{16}$$

To identify the optimum configuration of the data memory we consider the principal dimensions (*W* and *D*) and the data coding (*d*-of-*D*) to be fixed. To maximise the probability of correct recovery of all of the data (P_c) for a given number of entries (*Z*) we observe from equation 14 that this means minimizing h^w or, equivalently, $w \ln h$. Thus we seek a solution where

$$\frac{d}{dw}(w \ln h) = \ln h + \frac{w}{h} \cdot \frac{dh}{dw} = 0 \tag{17}$$

Equation 5 can be rewritten as:

$$w \cdot \frac{Z \cdot d}{W \cdot D} = -\ln(1 - h) \tag{18}$$

Differentiating with respect to *w* and substituting the value of *Z* from equation 5 yields:

$$\frac{dh}{dw} = -\frac{(1 - h) \cdot \ln(1 - h)}{w} \tag{19}$$

Substituting this result into equation 17 gives the following formula for *h*:

$$h \cdot \ln h - (1 - h) \cdot \ln(1 - h) = 0 \tag{20}$$

This has the solution $h = 0.5$, agreeing with previous results that show that an occupancy of 0.5 gives the maximum recoverable information content in a correlation matrix memory (Willshaw et al, 1969).

Substituting the expression for Z from equation 5 into equation 16 yields:

$$h^{-w} \cdot w \;=\; \frac{\ln(1-h)}{\ln(P_c)} \cdot W \cdot \frac{D}{d} \cdot (D-d) \tag{21}$$

Equation 21 can be used (with $h = 0.5$) to determine the optimum number of active address decoder words (w) for a given choice of data memory dimensions (W, D and d) and an acceptable probability of error-free recovery of all written data (P_c).

It can be argued, however, that totally error-free data recovery is too restrictive a requirement for this style of memory. An alternative definition of capacity is to establish the maximum number of data items that can be recovered correctly, allowing that other items may be recovered with errors. The expected number of correctly recoverable data words in the presence of errors is (from equation 13):

$$E_c \;=\; Z \cdot [1 - h^w]^{D-d} \tag{22}$$

For a given number of stored values (Z) we can vary the number of active address decoder rows (w) to maximize the expected number of correctly recoverable data words (E_c) and, as before, we find the maximum corresponds to occupancy $h = 0.5$.

If we continue writing entries until every new entry causes one or more additional read-out errors we have reached the maximum useful capacity of the data memory. This point is reached when $dE_c / dZ \;=\; 0$. Differentiating the natural logarithm of equation 22 gives:

$$\frac{1}{E_c} \cdot \frac{dE_c}{dZ} = \frac{1}{Z} - \frac{D-d}{1-h^w} \cdot \ln h \cdot h^w \cdot \frac{dw}{dZ} = 0 \tag{23}$$

At constant occupancy h equation 5 implies that wZ is constant, so $dw/dZ = -w/Z$. Substituting into equation 23 yields:

$$\frac{h^{-w} - 1}{w} = -(D - d) \cdot \ln h \tag{24}$$

This gives a value for the number of active address decoder rows (w) that is smaller than that given by equation 21, reflecting the fact that a higher w gives a memory that has fewer errors but lower capacity. A lower w results in more errors, but gives a higher capacity as each written entry sets fewer weights. In both cases, the maximum extractable information occurs when the occupancy $h = 0.5$.

Note that neither equation 22 nor equation 24 depends on the number of locations in the memory (W). This shows that the memory is linearly scalable; the maximum capacity is achieved when the occupancy $h = 0.5$, at which point the number of active address decoder rows, w, is given by Equation 24. Then, the proportion of values that are recoverable without error (E_c/Z) is given by Equation 22. Finally, the number of entries that must be written to achieve the maximum recoverable number of data values scales linearly with the number of locations, W (Equation 5). Hence the maximum recoverable data also scales linearly with W.

The behaviour of the memory is most clear when a typical plot of equation 22 is viewed. Such a plot is shown in 3-D in figure 3 and in contour form in figure 4. The example system has 4096 memory locations (W) and stores data using an 11-of-256 code; the choice of d-of-D code is discussed further in Section 5.2.

Referring to figure 4, the closed contours indicate how the expected number of correct words (E_c) varies with the number of stored entries (Z) and the number of active address decoder rows (w). The maximum is at $E_c = 5{,}332$, and the error rate there (the relative frequency of stored patterns

that are recovered with some error) is just over 11%. Note that this maximum capacity is greater than 4,096, the number of storage locations; this is possible because the data is sparse in both dimensions of the storage matrix. The central contour is at E_c = 5,000, and the contours then go down in steps of 500. The south face of the 'hill' is very nearly planar, with very few recovery errors (E_c is very close to Z), except at the west end where the number of active address decoder rows (w) is small. Performance falls off rapidly if either w is too small or Z is too large, as shown by the steep slopes on the west and north sides of the 'hill' respectively. If w is too small there is insufficient redundancy in the way the data is stored and even a low level of interference from other stored values causes errors; if Z is too large then the interference from other stored values is sufficient to cause errors whatever the level of redundancy.

Superimposed on the contours in figure 4 are two other sets of data. The hyperbolae show where the occupancy, h, has values 0.2 (to the south-west), 0.5 (centre) and 0.8 (to the north-east). Constant h lines correspond to constant wZ; see equation 5. Note how h = 0.5 passes directly through the summit of the 'hill'. On the south face are curves showing where the probability of error-free read-out (P_c) has values 0.1, 0.5 and 0.9 (from equation 14, with the 0.1 value higher up the slope, 0.5 in the middle, and 0.9 lower down). The maxima of these curves also lie on h = 0.5. Note how close together these curves are, indicating that the maximum capacity with a high probability of error-free read-out is relatively insensitive to the interpretation of 'high probability'.

As a final observation, note that the optimum value for the number of active address decoder rows (w) for capacity with errors (at the summit of the 'hill') is around 11, whereas the optimum value for error-free recovery (where P_c = 0.5 crosses h = 0.5) is around 20, and the capacity with errors is about twice the error-free capacity. Any practical use of the memory will operate some-

where in this range, the particular operating position being chosen to achieve an appropriate balance between capacity and error rate.

4.3. Address decoder analysis

The address decoder threshold *T* can be related to the number of active address decoder rows, *w*. The active address decoder rows include all those that have *T* or more 1s in common with the input address. From equation 1, the probability that a row has T or more bits in common with the input address is:

$$p_a = \frac{1}{C_a^A} \cdot \sum_{k=T}^{a} C_k^i \cdot C_{a-k}^{A-i} \tag{25}$$

The expected number of active address decoder rows is then:

$$\hat{w} = W \cdot p_a \tag{26}$$

The relationship between the number of active address decoder rows (*w*), the number of 1s in each address decoder row (*a*) and the address decoder threshold (*T*) is shown in figure 5 for a memory with *W* = 4096 locations using an 11-of-256 input address. The useful part of this function is plotted in more detail in figure 6. This shows that the optimum number of active address decoder rows can be achieved by using one of several different combinations of the address decoder threshold and the number of 1s in each address decoder row.

If the memory is operating with error-free input the same address decoders will always fire, but the number that fire is subject to statistical variation from one input address to another. The *W* independent address decoder rows result in a binomial distribution for *w* with mean given by equation 26 and variance given by:

$$\sigma_a^2 = W \cdot p_a \cdot (1 - p_a) \tag{27}$$

In particular, the probability of *w* taking a particular value *w'* is given by:

$$p_{w'} = (1 - p_a)^{W - w'} \cdot p_a{}^{w'} \cdot C^{W}_{w'} \tag{28}$$

Equation 14, which gives the probability that all of the stored data values can be recovered without error, now becomes:

$$P_c = \prod_{w'} [1 - h^{w'}]^{Z \cdot p_{w'} \cdot (D - d)} \tag{29}$$

Similarly equation 22, which gives the expected number of correctly recoverable data items (ignoring those recovered with error) becomes:

$$E_c = \sum_{w'} p_{w'} \cdot Z \cdot [1 - h^{w'}]^{D - d} \tag{30}$$

The result of the statistical variation of the number of active address decoder rows (*w*) is illustrated in figure 7, which shows the performance of the same memory as used for figure 4 but with the spread now taken into account. The summit is lower, at *Ec* = 4,445, where the number of stored values *Z* = 5,440 and *w* = 15; it has also moved off the occupancy *h* = 0.5 curve, and corresponds to *h* = 0.575. The error-free performance is poor at low *w*, but the general shape of the 'hill' is unaffected. It is clear, though, that for optimal performance (however defined) the address decoder threshold (*T*) must be set to give a mean value of *w* somewhat higher than that which would be chosen without taking the spread into account.

It might be thought that the variation of the number of active address decoder rows (*w*) could be reduced (or even eliminated) by imposing a '*w*-max' mechanism similar to that used to select the '*d*-max' data outputs. However, as noted at the end of Section 2, the activation of the address decoder rows is quantized and the *w* active outputs are dominated by decoders that exactly meet the threshold. It is not possible, therefore, to discriminate between the outputs that fire to select the

same number each time, and some spread in *w* must be accepted. The statistical independence of the individual decoders inevitably leads to the binomial distribution used above.

4.4. Numerical simulations

Numerical simulations were carried out to confirm the validity of the analytical results shown in figure 7. This process is not as straightforward as might at first be assumed, since the expected number of active address decoders $\hat{w}$ is hard to control smoothly over the desired range as it is a rather complex function of the address decoder parameters (see equation 25). The number of stored values (*Z*) is straightforward to control, but the desired coverage of the $\hat{w}$ values was achieved by selecting rather *ad hoc* combinations of values for the number of 1s in the input address (*i*), each address decoder row (*a*) and the address decoder threshold (*T*). The contour-plotting software required a uniform distribution of points along the horizontal axis, and this was obtained by linear interpolation of the results computed by a single simulation at each of the ($\hat{w}$, *Z*) points. A similar procedure was used later to produce figure 12; Section 5.1 includes more detail on the precise data points used there.

The numerical simulations of the error-free memory are shown in figure 8 and confirm the analytical results presented in figure 7. The differences between the figures are mainly due to interpolating the contour values from the coarse grid used for the base simulation cases, the coarseness of the grid being itself an unavoidable consequence of the highly quantized behaviour of the address decoders noted above.

5. Error recovery

Now we consider the case where the memory has been written with correct address and data values, but an attempt is made to recover a data value with a corrupt address.

In the simplest case, the corrupt address is a valid *i*-of-*A* code but it has a number (*n*) of bits set in incorrect positions; that is, some of the bits that should be 1 are 0, and an equal number that should be 0 are 1. The effect of this will be to cause some of the address decoder rows that were active during the write operation to be inactive during the corresponding read operation, and conversely some that were inactive to be active. We denote the number of address decoder rows that are active during both read and write operations by w_s (*s* indicating 'signal'), and the number that are active only during read by w_n (*n* denoting 'noise'). It is clear that these two parameters fully characterize the operation of the memory under noisy input conditions.

On the basis that most of the active address decoder rows exactly meet the threshold during the write operation, they will also be active during a read operation with the address input containing errors provided that those errors do not coincide with any of the address input bits that contribute to meeting that threshold. To a first approximation the number of address decoders that are active both during write and during read with an *n*-bit input error is therefore given by:

$$w_s = \left(1 - \frac{T}{i}\right)^n \cdot w \tag{31}$$

Numerical simulation results illustrating this dependency are shown in figure 9, where w_s / w is plotted against the address decoder threshold (*T*) for 1 to 5 input bit errors using an 11-of-256 input code to the address decoder. (These results agree reasonably well with equation 31 for small values of *T*, but the approximation is poor for values of *T* greater than 3.) The best performance, with the highest value for w_s / w, is obtained by selecting a small value for *T*. Keeping the number of active address decoder rows (*w*) close to an optimal value requires a small value to be chosen for the number of 1s in each address decoder row, *a* (see figure 6). This is why it is desirable to be able to choose *a* independently from the number of 1s in each input address, *i*.

The correct address decoder rows active during a read operation (numbering w_s) will be a subset of the w that fired in the corresponding write operation and the undesired active address decoder rows (numbering w_n) will be a subset of the $W - w$ that were inactive during the write operation. Under an n-bit input error, to a first approximation w_n is given by:

$$w_n = \left(1 - \left(1 - \frac{T}{i}\right)^n\right) \cdot \hat{w} \tag{32}$$

Note that, unlike w_s which is proportional to w, to a first approximation w_n is independent of w. Thus a data value that is written to a small number of hard locations will be disproportionately affected by address-input noise.

In the presence of more input errors the proportion of undesired active address decoder rows will increase, but the performance of the memory is still fully determined by the variables w_s and w_n.

The expected activation level of a data output that was written as a 1 (formed as the sum of those elements of the data memory matrix column that connect to an active word line) is now:

$$E(X_1) = w_s + w_n \cdot h \tag{33}$$

The expected activation level of a data output that was written as a 0 is:

$$E(X_0) = (w_s + w_n) \cdot h \tag{34}$$

Both activation levels are subject to statistical variation following binomial distributions. The respective variances are given by:

$$\sigma_1^{\,2} = w_n \cdot h \cdot (1 - h) \tag{35}$$

$$\sigma_0^{\,2} = (w_s + w_n) \cdot h \cdot (1 - h) \tag{36}$$

The data memory output distribution is bimodal, having d outputs with expected value $E(X_1)$ and $D - d$ outputs with expected value $E(X_0)$. The d-of-D code output from the data memory selects the highest scoring d outputs. The threshold for selection is the level, X_t, where the same number of X_0 outputs exceed X_t as X_1 outputs fall below X_t. That is, the expected number of erroneous 'noise' data outputs, d_n, is given by:

$$E(d_n) = d \cdot p(X_1 \leq X_t) = (D - d) \cdot p(X_0 \geq X_t) \tag{37}$$

This is sufficient to determine X_t and, hence, $E(d_n)$. The convergence requirement for an auto-associative memory (where the a-of-A and d-of-D codes are the same) is that the output errors are fewer than the input errors, $E(d_n) < a_n$. Where the input and output codes differ the convergence requirement is harder to define, but should probably be based on a consideration of the information content of the input and output codes.

An example plot of the (binomial) output distributions for X_0 and X_1 is shown in figure 10. Here the output d-of-D code is 11-of-256, $w_s = 20$, $w_n = 10$, and $h = 0.5$. The plot shows the frequency distribution of activation levels in the X_0 ('o') and X_1 ('+') outputs. Although the distributions do overlap, the expected number of output errors is less than 1 in this case (in fact, it is close to 0.6). As the occupancy h increases the mean of the X_0 distribution (equation 34) moves to the right, the overlap between the distributions increases, and the output error rate rises.

The optimization of the error-recovery properties of the memory is complex. The ratio of w_s to w_n is optimised by the choice of address decoder parameters, but it is easy to increase them both proportionately. This will have the effect of reducing the ratio of variance to mean of both distributions (equations 35 and 36) and thereby reducing the overlap. However, it will also cause h to increase (equation 3), thereby moving the distributions closer together and increasing the overlap.

To estimate the overall effect of increasing w_s and w_n, let us assume that they are both scaled by a factor *k*. The difference between the distribution means, from equations 33 and 34, is:

$$E(X_1) - E(X_0) = w_s \cdot (1 - h) \tag{38}$$

Both variances scale similarly by a factor of *k*, and the ratio of the difference between the means to the standard deviation gives an estimate of the distribution overlap:

$$\frac{E(X_1) - E(X_0)}{\sigma_i} \propto \sqrt{\frac{k(1-h)}{h}} \tag{39}$$

where $i = 0$ or 1. Now equation 5 can be rewritten:

$$h = 1 - e^{-Z\left(\frac{wd}{WD}\right)} \tag{40}$$

With this, and noting that *w* is proportional to *k*, we obtain:

$$\frac{E(X_1) - E(X_0)}{\sigma_i} \propto \sqrt{\frac{k}{e^{Ak} - 1}} \tag{41}$$

where $A = Z \cdot (w_0 d / (WD))$ and w_0 is some baseline value of *w*. Equation 41 is a monotonic decreasing function of *k* for positive *k* values, showing that increasing *w* has a generally adverse effect on the error performance of the memory.

Clearly this result should not be taken as indicating that *w* should be made as small as possible. All the factors considered in the error-free memory still apply. It simply demonstrates, perhaps counter-intuitively, that increasing the redundancy under error conditions may have no net benefit as the gains are more than outweighed by the concomitant increase in occupancy.

5.1. Numerical simulations

A range of memories with different address decoder parameters was simulated numerically, under single-bit input error conditions. As with figure 8 (see Section 4.4), the range of values for the

expected number of active address decoders $\hat{w}$ was achieved by simulating memories with a range of values for the number of 1s in each address decoder row, *a*, and a range of address decoder thresholds, *T*. The values used are shown in figure 11, which is a partial confirmation by numerical simulation of figure 6. For each of these configurations the memory was simulated with the number of stored values (*Z*) equal to 100, 250, then in increments of 250 to 4,000, 4,500, 5,000 and then in increments of 1,000 to 10,000. The contour plotting program required a uniform horizontal distribution of input points, and this was again obtained by linear interpolation.

The results, shown in figure 12, confirm that the optimum memory configuration operates with the number of active address decoder rows *w* very close to the optimal values for the error-free memory. This is similar to the demonstration by Kristoferson that varying the number of active address decoder rows between read and write operations has no benefit in improving error tolerance (Kristoferson, 2001).

The maximum number of recoverable error-free values E_c appears to be about 4,300 at $w = 14$ and with the number of stored values, *Z*, being about 5,400. The loss of useful capacity, from 4,445 in the error-free case to 4,300 with single-bit input errors (which is one bit in eleven) is, perhaps, lower than might be expected.

5.2. Data error recovery

We can further extend the tolerance of the memory to errors if we use a subset of the full data *d*-of-*D* space as valid symbols, resolving each output to the nearest valid symbol. For example, we could allow *f*-bit error recovery by mapping every *d*-of-*D* code with at least $d - f$ 1s in common with a valid symbol to that symbol. With such a mapping there must be fewer valid symbols and the information content of each symbol is diminished accordingly. Since the great majority of the mapped symbols will be at the maximum distance from the valid symbol, equation 6 becomes:

$$I(Z) = Z \cdot \log_2[C_d^D / (C_f^d \cdot C_f^{D-d})] \text{ bits} \tag{42}$$

Now to a first approximation the value of Z that gives maximum capacity varies as $1/d$ (see equation 5; the optimum number of active address decoders, w, depends only very weakly on d – see equation 24), so we can identify the d-of-D code that gives the highest information content under assumed error conditions. We wish to solve:

$$\frac{d}{dd}\left\{\frac{1}{d} \cdot \log_2[C_d^D / (C_f^d \cdot C_f^{D-d})]\right\} = 0 \tag{43}$$

The combinatorial terms can be approximated closely using:

$$\ln(x!) = \frac{1}{2}\ln(2\pi x) + x\ln x - x \tag{44}$$

Carrying out the differentiation yields, for $d << D$:

$$d = f(\ln(D \cdot (d-f)) - 1) - 2\ln(f!) + \ln d + \frac{1}{2}\ln\left(\frac{2\pi}{d-f}\right) - 1 + \frac{d}{2(d-f)} - \frac{(d^2 + f^2)}{2D} \tag{45}$$

Numerical solutions of equation 43 for a 256-bit code with 1-, 2-, 3- and 4-bit errors are d = 8, 14, 20 and 25 respectively. Equation 45 gives the same result for these cases, and it also serves to indicate the general trend of the solution which has $d = f \cdot \ln D$ as its dominant term.

Throughout this paper we have been concerned only with maximizing the number of data values recovered without any error, so we have treated a single-bit error as invalidating a complete output value. Many of the erroneous outputs that occur at the summit of the hill, 20% in figure 12 for example, will have only single-bit errors and therefore contain significant information that we have neglected but that could be recovered using a suitable nearest-valid-symbol scheme as described above.

The choice of which *d*-of-*D* code should be employed in any application is a matter of determining the required number of unique codes and the required resilience to errors. The 11-of-256 code used for the examples throughout this paper is an arbitrary choice, with no identified properties that make it optimal in any respect.

6. Memory efficiency

We now return to the issue of the efficiency of the *N*-of-*M* memory operating without data error correction. Referring back to equation 7, we can base the efficiency measure on the recoverable data stored:

$$\eta(E_c) = \frac{I(E_c)}{D \cdot W} = \frac{E_c}{D \cdot W} \cdot \log_2[C_d^D] \tag{46}$$

With an 11-of-256 data code, each stored value contains 62 bits of information. For the example discussed earlier, with no address errors 4,445 values can be recovered without error (from a total of 5,440 values written into the memory) from the 4,096 word memory (that is, $W = 4{,}096$). This gives an efficiency of 0.26 bits of information per bit of writeable data store.

Palm and Sommer (1996) presented an extensive analysis the efficiency of various configurations of associative binary network operating under similar (though not identical) conditions to the data store analysed here. They demonstrated an asymptotic efficiency of 0.69 bits of information per synapse for very large, sparse binary networks with error-free addressing and the same Hebbian learning algorithm that we have employed. The efficiency of finite memories of a size comparable to ours is significantly lower, in a similar range to the 0.26 bits per synapse we have found, though direct comparison is not straightforward as their definition of capacity is different from ours.

7. Neural implementation

Kanerva's sparse distributed memory has been implemented directly in hardware in various forms (Flynn et al, 1988), but these implementations have required dedicated units for the address decoder Hamming distance computations and the data memory matrix of up-down counters. In contrast, the *N*-of-*M* sparse distributed memory can readily be implemented using conventional leaky integrate-and-fire spiking neurons with binary synaptic weights and a local Hebbian learning algorithm. The 'leaky' property is only required here in order that all neurons in the system will settle down into inactive states between 'waves' of activity. This section gives a brief overview of the operation of such an implementation.

The memory is activated when an *i*-of-*A* address input pattern arrives. This means that spikes arrive on exactly *i* of the *A* input neurons at (roughly) the same time. (We are also interested in an extension to *N*-of-*M* coding where the order of arrival carries additional information – a form of *rank order coding* (Thorpe et al, 2001) – but that is not the subject of this paper.)

The address decoder (see figure 2) comprises *W* neurons. The connections and weights between the input neurons and the address decoder neurons are fixed, as are the thresholds of the address decoder neurons. One can view the connection matrix as being complete – that is, every input connects to every address decoder neuron – and the weights are such that each address decoder neuron has *a* weights set to 1 and $A - a$ weights set to 0. Alternatively, since a 0 weight is equivalent to no connection, one can view each address decoder neuron as being connected to just *a* of the input neurons. In either case, the *a* 1s or input connections are selected at random for each address decoder neuron.

With this configuration, and with an appropriate threshold set for the address decoder neurons, firing *i*-of-*A* inputs will cause *w*-of-*W* address decoders to fire, with *w* subject to some variation as described in Section 4.3.

The data memory comprises *D* neurons each of which has all *W* address decoder neurons as inputs. In addition, each data neuron has a unique write data input. Initially, all of the synaptic connections from the address decoder neurons have weights set to 0. The synaptic connections from the write data inputs have weights whose numerical strength is irrelevant, but whenever a write data input neuron fires it must force its respective data memory neuron into a Hebbian learning mode that persists throughout the associated incident wave of firing address decoder neurons.

A write cycle now proceeds as follows. Firstly, the *d*-of-*D* write data input is signalled by *d* of the write data neurons firing, in turn putting *d* of the data memory neurons into Hebbian learning mode. Then the address inputs fire, causing *w* address decoders to fire. Wherever an address decoder output spike meets a data memory neuron that is in Hebbian learning mode the corresponding synaptic connection is set to 1. Once the wave of incident address decoder neuron spikes has passed, the ‘leaky’ aspect of all of the neurons causes the activation levels of all neurons to settle back to zero and the data memory neurons cease to be in Hebbian learning mode. After a suitable delay the memory is ready for its next read or write cycle.

(It could be argued that it is more biologically plausible for the address decoder inputs to fire before rather than after the data inputs. The write cycle could operate this way, but then each data neuron would have to record each firing address decoder input, requiring each synapse rather than just each neuron to be able to perform a short-term memory function.)

The write data inputs are inactive during a read cycle. The read operation is initiated by the address inputs firing which, in turn, cause *w* address decoder neurons to fire. These then cause the

activation levels on the data memory neurons to rise (assuming that previous write cycles have set at least some of the relevant synaptic weights). A generalised *winner-takes-all* (*d*-WTA) style of output logic selects the *d* data memory neurons with the highest activation levels and these issue spikes which encode the data output. Again, a suitable delay is required to allow all neurons to settle back into their idle states before the memory is ready for its next read or write cycle.

Here we do not provide details of the implementation of the *d*-WTA neural circuit that is required to generate the *d*-of-*D* output code, but merely note that such circuits, generally based on laterally connected inhibitory neurons that discourage other neurons from firing once the required number of outputs has fired, have been studied extensively in the past (Maass, 2000).

8. Summary and conclusions

An architecture for a sparse distributed memory based upon the use of sparse binary *N*-of-*M* codes has been presented. Theoretical analysis of the memory allows its performance to be predicted and optimised by careful selection of its parameters. Numerical simulations confirm the validity of the theoretical analysis.

The memory is well-suited to implementation using spiking neurons, requiring only unipolar binary synaptic weights with a simple Hebbian learning algorithm and yielding a constant, low, average neural activity level.

Our *N*-of-*M* memory may have relevance to the understanding of biological neural systems; for example, activity in the brain is sparse and distributed, as is the activity in this memory. In addition, it has been observed that the cerebellum has an organization similar to Kanerva's memory, but the 'address decoder' neurons in the cerebellum (the granule cells) have very few synaptic inputs – typically 3 to 5 (Marr, 1969) – whereas Kanerva's memory requires address decoders with very large numbers of inputs. In our analysis, we observed that noise immunity is optimised by using an ad-

dress decoder with a low threshold (equations 31 and 32) and an *a*-of-*A* code with a low value for *a*, in the same range as that observed in the cerebellum. This property is shared by the 'hyperplane' variant of Kanerva's memory (Jaeckel, 1989) to which our memory is closely related. A low *a* value also minimizes cost in an implementation where each synaptic input has to be grown.

The *N*-of-*M* memory shows many similarities to Marr's description of the cerebellar cortex (Marr, 1969), the major difference being the absence of the inhibitory interneurons that Marr surmises regulate activity. Such regulation is implicit when *N*-of-*M* codes are used.

The memory combines features of the 'hyperplane' variant of Kanerva's memory with the storage efficiency of a binary correlation matrix memory to yield a design that can readily be implemented using spiking neurons and a simple Hebbian learning rule. The analysis presented in this paper enables an implementation of the memory to be optimised for a particular application and provides insights into its likely performance and characteristic modes of failure.

Acknowledgements

The authors gratefully acknowledge the support of EPSRC through ROPA grant GR/M46396 (which supported WJB) and funding from Cogniscience Ltd (which supported JMC and ST). The use of *N*-of-*M* codes in this work was strongly influenced by Simon Davidson's PhD thesis (Davidson, 1999), and the authors are grateful to Peter Ivey for drawing this thesis to their attention. We are also grateful to Jim Austin for his helpful comments on an early draft of this paper, and to the journal referees who were exceptionally thorough and helpful in their comments.

References

Aleksander, I. and Stonham, T.J. (1979). Guide to Pattern Recognition using Random-Access Memories. *Computer and Digital Techniques*, **2**, 29-40.

Austin, J. (1995). Associative Memories and the Application of Neural Networks to Vision. In R. Beale (Ed.), *Handbook of Neural Computation*. Oxford University Press.

Austin, J. and Stonham, T.J. (1987). An Associative Memory for Use in Image Recognition and Occlusion Analysis. *Image and Vision Computing*, **5**, 251-261.

Casasent, D. and Telfer, B. (1992). High Capacity Pattern Recognition Associative Processors. *Neural Networks*, **5**, 687-698.

Davidson, S. (1999). *On the Application of Neural Networks to Symbolic Systems***.** PhD Thesis, University of Sheffield, UK.

Feller, W. (1950). *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, New York, 33-39.

Flynn, M.J., Kanerva, P., Ahanin, B., Bhadkamkar, N. Flaherty, P. and Hickey, P. (1988). *Sparse Distributed Memory Prototype: Principles of Operation*. Technical Report CSL-TR-87-338, Computer Systems Laboratory, Stanford University.

Jaeckel, L.A. (1989). *A Class of Designs for a Sparse Distributed Memory.* RIACS Technical Report 89.30, NASA Ames Research Centre.

Kanerva, P. (1988). *Sparse Distributed Memory.* Cambridge MA: MIT Press.

Kanerva, P. (1993). Sparse Distributed Memory and Related Models. In Hassoun, M.H. (ed.). *Associative Neural Memories: Theory and Implementation*. Oxford University Press.

Kohonen, T. (1972). Correlation Matrix Memories. *IEEE Transactions on Computers*, **C-21**, 353-359.

Kristoferson, J. (2001). Some Results on Activation and Scaling of Sparse Distributed Memory. In Uesaka, Y., Kanerva, P. and Asoh, H. (eds.). *Foundations of Real-World Intelligence*. Stanford: CSLI Publications, 283-289.

Lomas, D. (1996). *Improving Automated Postal Address Recognition.* MSc dissertation, University of York, UK.

Maass, W. (2000). On the Computational Power of Winner-Take-All. *Neural Computation,* **12**, 2519-2535.

Marr, D. (1969). A Theory of Cerebellar Cortex. *Journal of Physiology,* **202**, 437-470.

Nadal, J-P. and Toulouse, G. (1990). Information Storage in Sparsely Coded Memory Nets. *Network,* **1**, 61-74.

Palm, G. and Sommer, F.T. (1996). Associative Data Storage and Retrieval in Neural Networks. In Domany, E., van Hemmen, J.L. and Schulten, K. (eds.). *Models of Neural Networks III: Association, Generalization and Representation*. New York: Springer-Verlag.

Schwenker, F., Sommer, F.T. and Palm, G. (1996). Iterative Retrieval of Sparsely Coded Associative Memory Patterns. *Neural Networks,* **9**, 445-455.

Thorpe, S., Delorme, A. and Van Rullen, R. (2001). Spike-based Strategies for Rapid Processing. *Neural Networks*, **14**, 715-725.

Turner, M. and Austin, J. (1997). Matching Performance of Binary Correlation Matrix Memories. *Neural Networks*, **10**, 1637-1648.

Verhoeff, T. (1998). Delay-Insensitive Codes – an Overview. *Distributed Computing,* **3**, 1-8.

Willshaw, D. J., Buneman, O.P. and Longuet-Higgins, H.C. (1969). Non-Holographic Associative Memory. *Nature,* **222**, 960-962.

Figure captions.

Figure 1: The hypergeometric probability distributions for the number of 'hits' (1s in matching positions) between a pair of random *N*-of-*M* codes for various values of *N*, and *M*=1000.

Figure 2: A sparse distributed memory using *N*-of-*M* codes.

Figure 3: The expected number of correctly recovered data items (E_c) as a function of the number of stored items (*Z*) and the number of locations each data item is stored in (*w*). The data memory has 4096 locations and the data uses an 11-of-256 encoding.

Figure 4: The expected number of correctly recovered data items (E_c), the occupancy (*h*) and the probability of all recovered data items being error-free (P_c), as functions of the number of stored items (*Z*) and the number of locations each data item is stored in (*w*). The data memory has 4096 locations and the data uses an 11-of-256 encoding. The maximum value of E_c is 5332, occurring at *h* = 0.5.

Figure 5: The relationship between the address decoder threshold (*T*), the number of 1s in each address decoder (*a*) and the number of address decoder rows that fire (*w*). The input address uses an 11-of-256 code and the memory has 4096 hard locations (*W* = 4096).

Figure 6: Detail view of the area of figure 5 near to the origin.

Figure 7: The expected number of correctly recovered data items (E_c), the occupancy (*h*) and the probability of all recovered data items being error-free (P_c) as functions of the number of stored items (*Z*) and the mean number of locations each data item is stored in (*w*). The data memory has 4096 locations and the data uses an 11-of-256 encoding. This differs from figure 4 in that allowance is made here for the variation in *w* resulting from the statistical properties of the address decoder. The maximum value of E_c is reduced to 4445 and now occurs at *h* = 0.575.

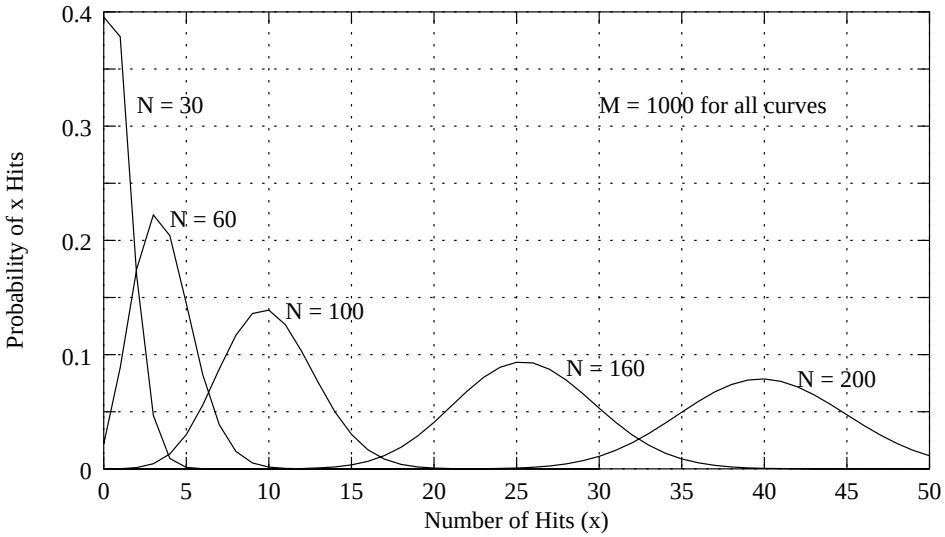
Figure 8: The results of numerical simulations of the memory for which analytical results were presented in figure 7. The correspondence with the analytical results is very close, the differences being due to the coarse grid used for the base simulations from which the contours were calculated by interpolation.

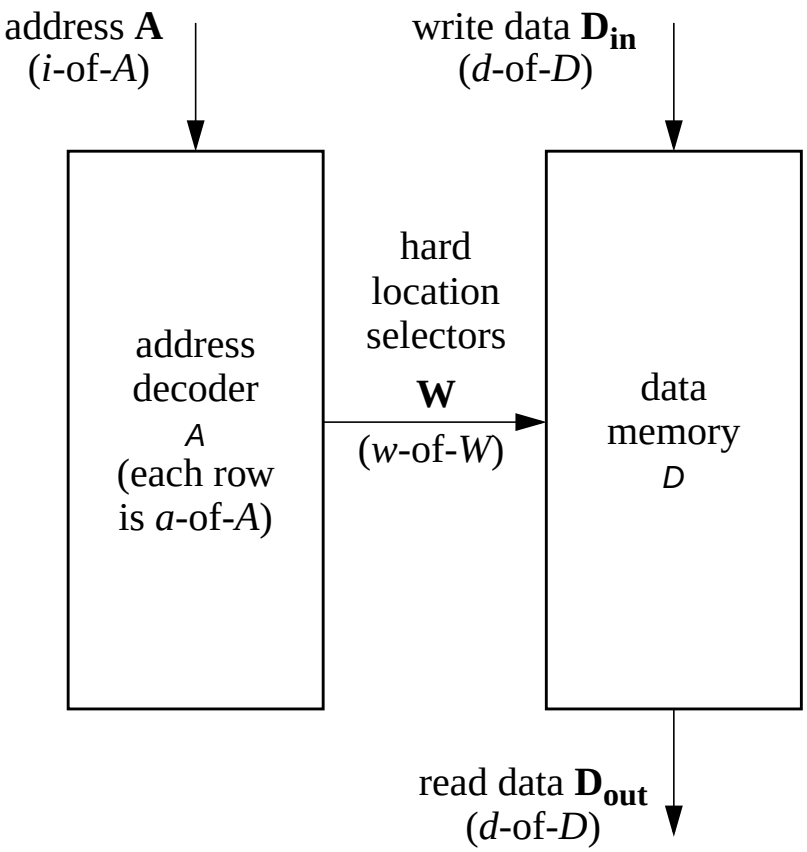
Figure 9: Effect of variations of the address decoder threshold (*T*) on the proportion of address decoders that fire in both the write and the read phases ($w_s / \hat{w}$), for various numbers of error bits (*n*) in the read address. For each value of *T* two adjacent values for the number of 1s in the address decoder (*a*) were chosen to give values of $\hat{w}$ above and below 30, and the results interpolated to $\hat{w} = 30$. The lower value of *a* for *T* = 2...11 was 3, 10, 20, 33, 48, 65, 85, 107, 132, 165.

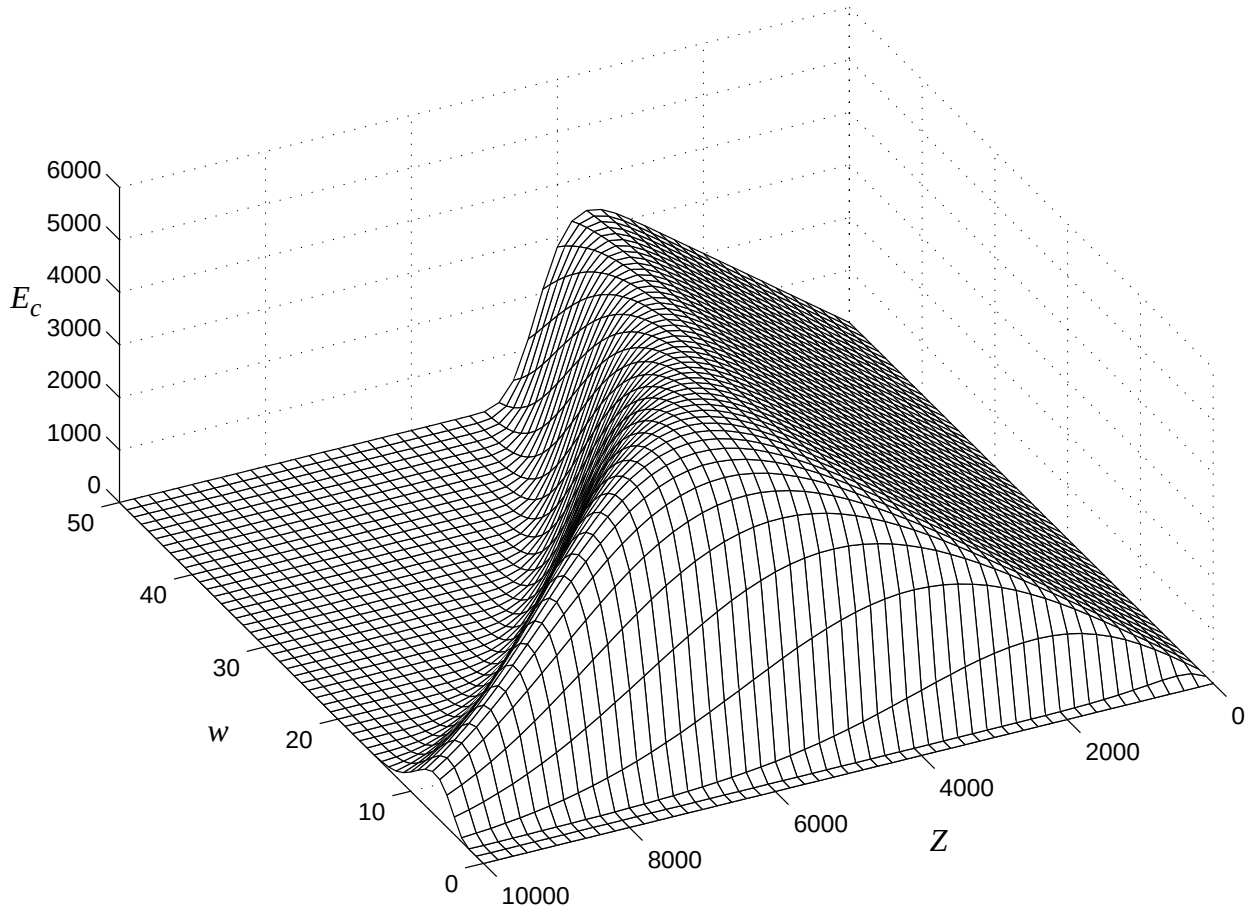
Figure 10: Example distributions for the activation level of a data output written as 0 (X_0 - "o"), and the activation level of a data output written as 1 (X_1 - "+"). Output errors will occur where these two distributions have significant overlap.

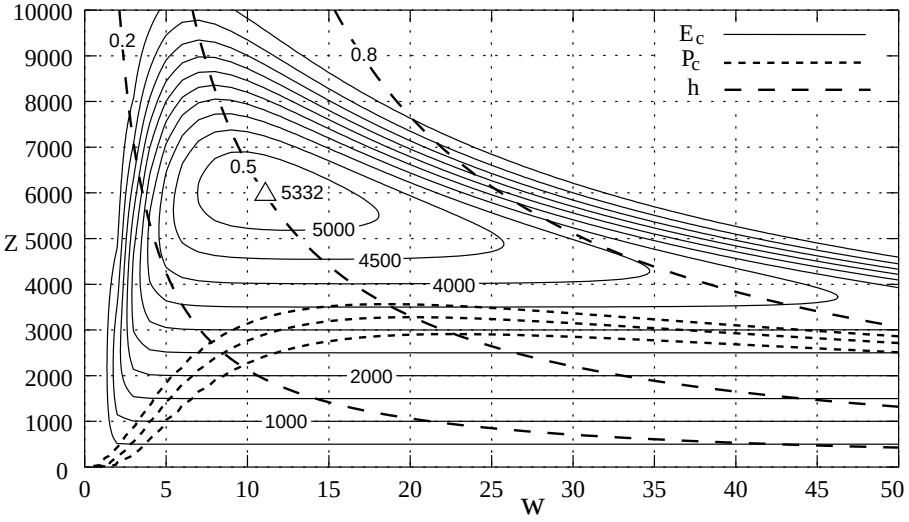
Figure 11: The address decoder configurations – the number of 1s in each address decoder row (*a*) and the address decoder threshold (*T*) – used in the numerical simulations to produce the values used in figure 12 for the mean number of active address decoder rows ($\hat{w}$). The address decoder has 4096 rows and the input address uses an 11-of-256 encoding in every case.

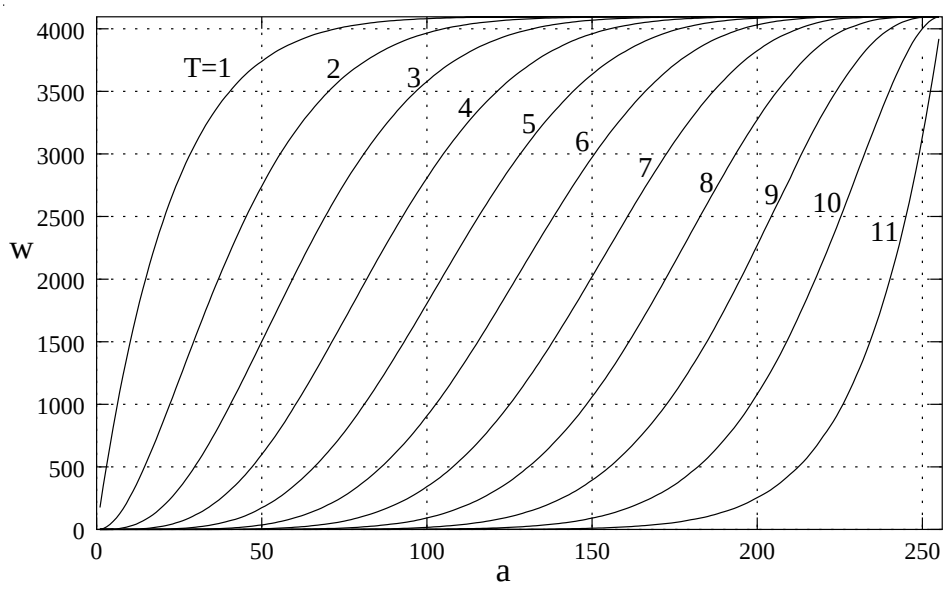
Figure 12: The performance of the memory with single-bit input errors. The expected number of correctly recovered data items (E_c) and the occupancy (*h*) are plotted as functions of the number of stored items (*Z*) and the mean number of locations each data item is stored in ($\hat{w}$). The data memory has 4096 locations and the data uses an 11-of-256 encoding.

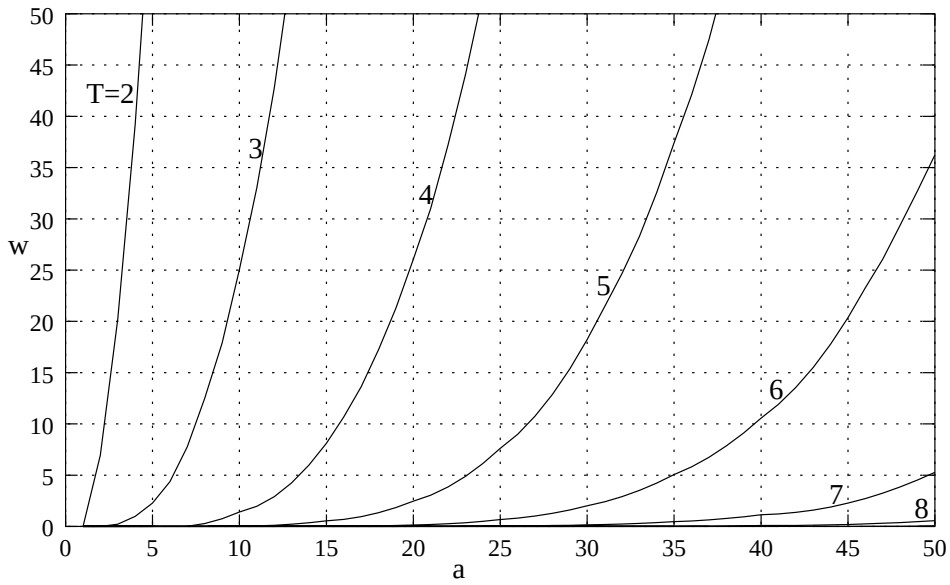

0.4
0.3
0.2
0.1
0
Probability of x Hits
N = 30
N = 60
N = 100
N = 160
N = 200
M = 1000 for all curves
0
5
10
15
20
25
30
35
40
45
50
Number of Hits (x)

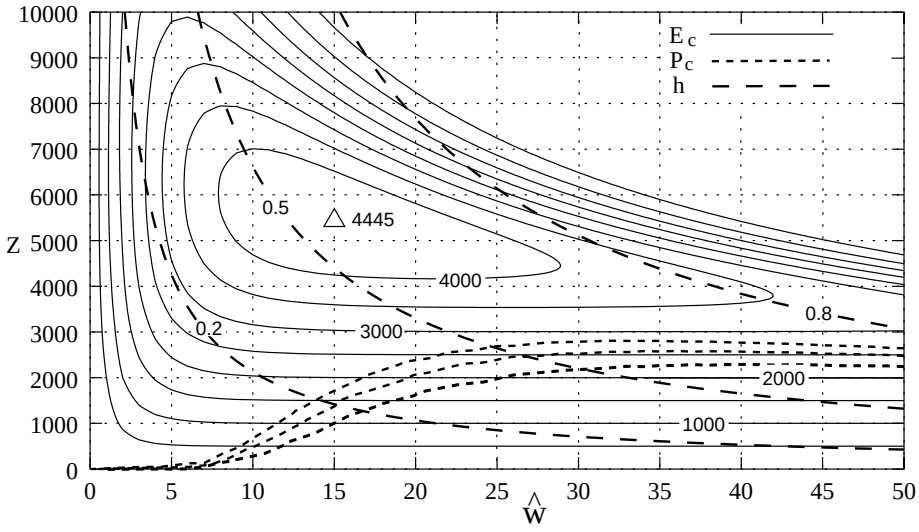

address **A**
(*i*-of-*A*)
write data $\mathbf{D_{in}}$
(*d*-of-*D*)
address
decoder
A
(each row
is *a*-of-*A*)
hard
location
selectors
W
(*w*-of-*W*)
data
memory
D
read data $\mathbf{D_{out}}$
(*d*-of-*D*)

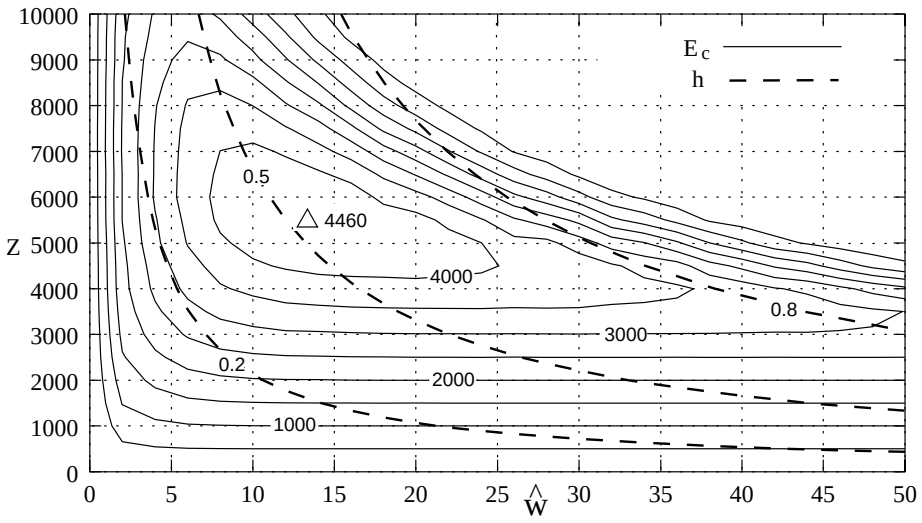

6000
5000
4000
3000
2000
1000
0
E_c
50
40
30
20
10
0
w
0
2000
4000
6000
8000
10000
Z

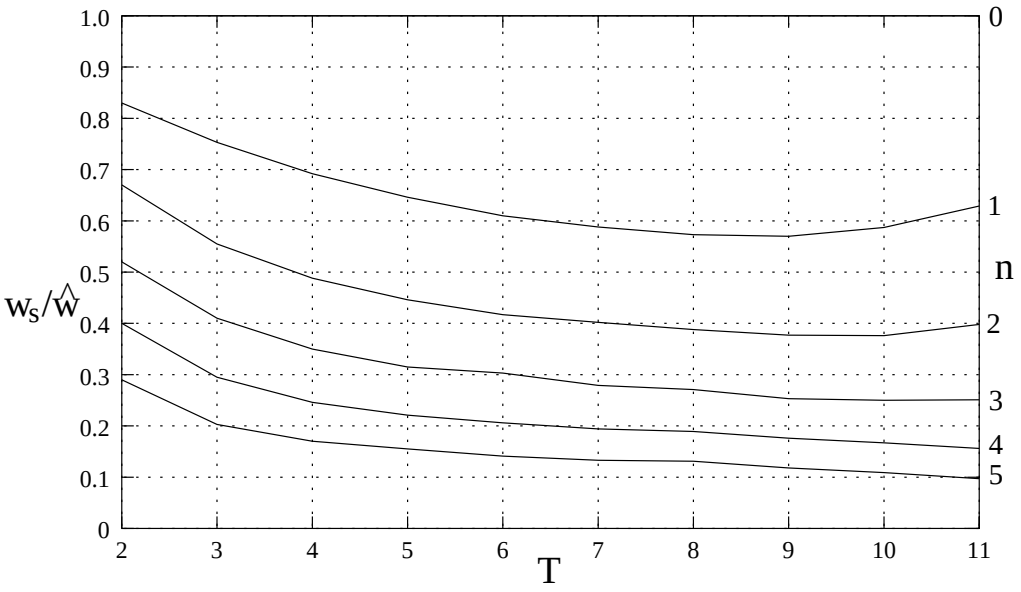

10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Z
0
5
10
15
20
25
30
35
40
45
50
W
0.2
0.8
0.5
5332
5000
4500
4000
2000
1000
Ec
Pc
h

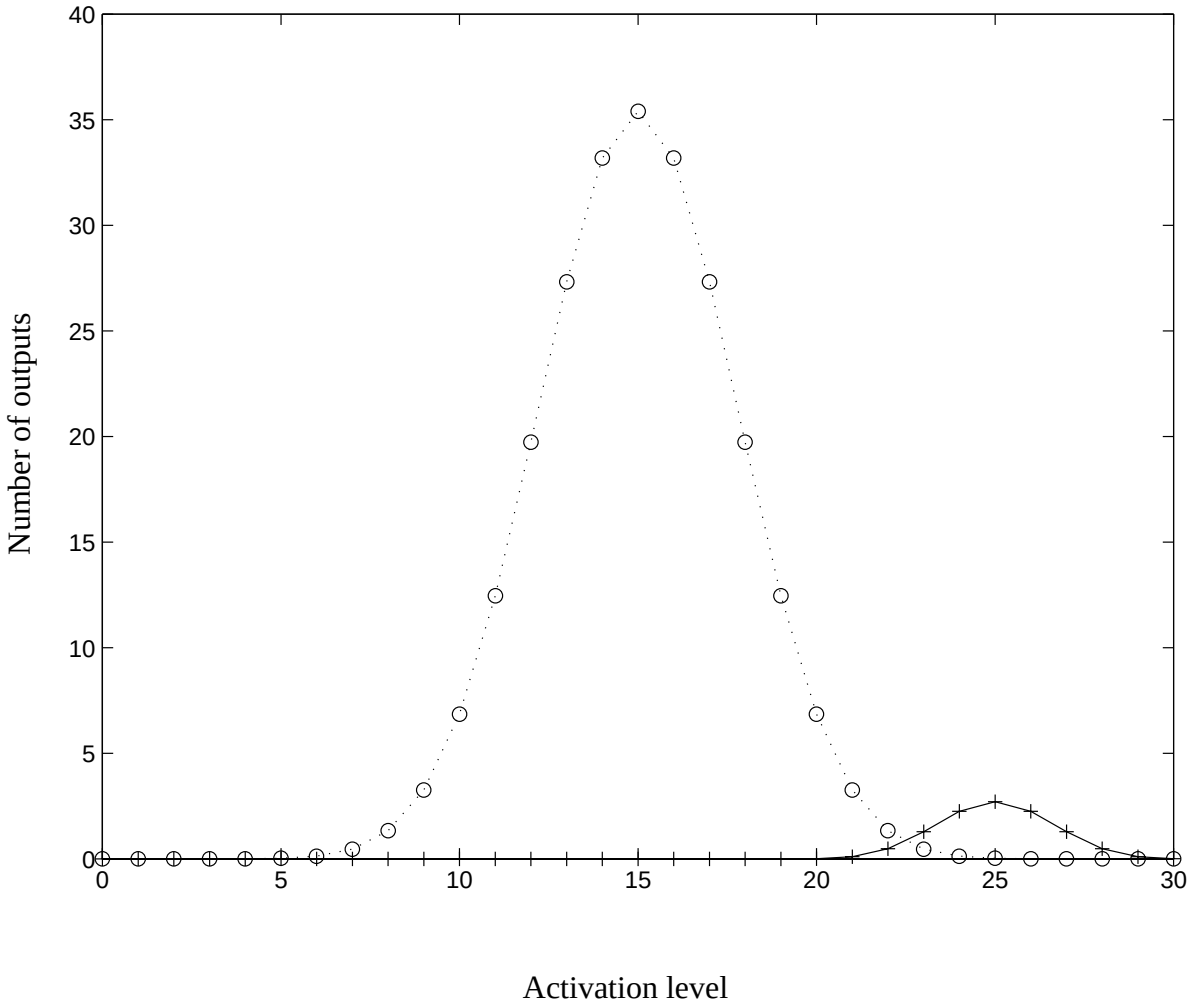

4000
3500
3000
2500
2000
1500
1000
500
0
w
0
50
100
150
200
250
a
T=1
2
3
4
5
6
7
8
9
10
11

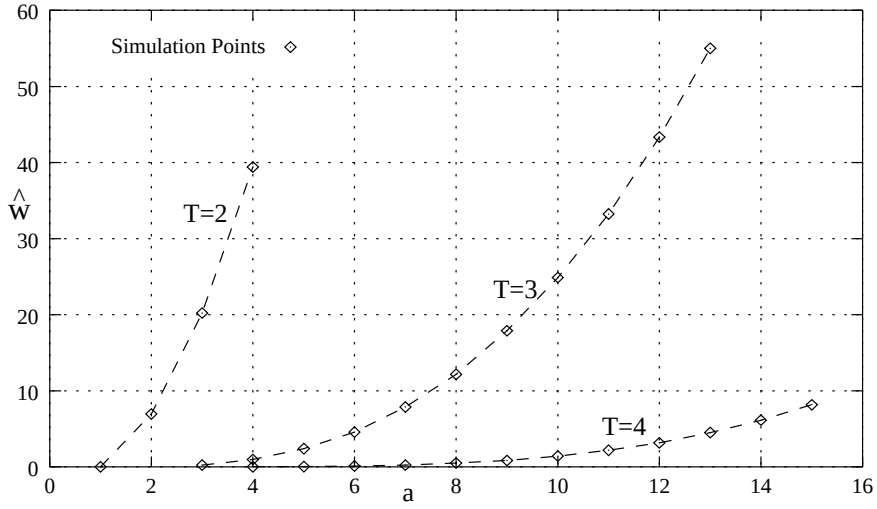

T=2
3
4
5
6
7
8
w
a
0
5
10
15
20
25
30
35
40
45
50

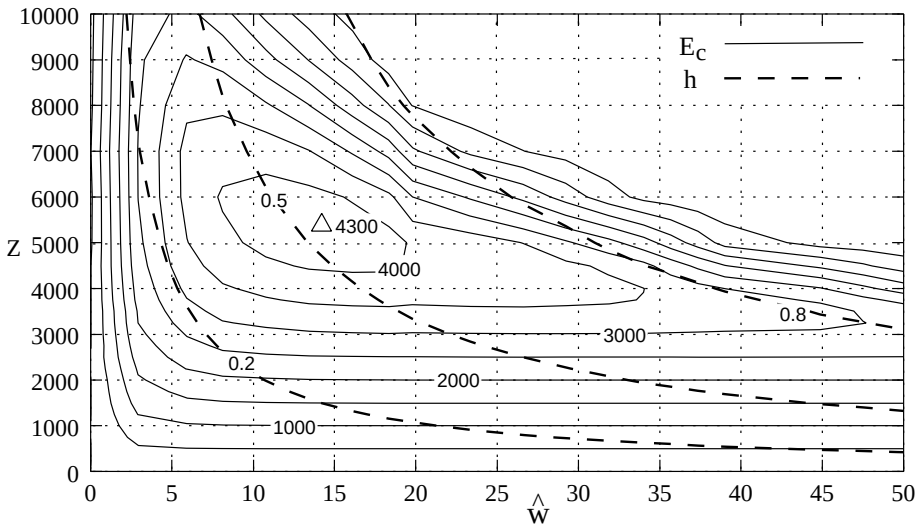

10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Z
0
5
10
15
20
25
30
35
40
45
50
W
E c
P c
h
0.5
4445
4000
0.2
3000
0.8
2000
1000


10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Z
0
5
10
15
20
25
30
35
40
45
50
W
E c
h
0.5
4460
4000
0.8
3000
0.2
2000
1000


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
9
10
11
T
$w_s/\hat{w}$
0
1
2
3
4
5
n


Number of outputs
Activation level
0
5
10
15
20
25
30
35
40


Simulation Points
T=2
T=3
T=4
W
a
0
10
20
30
40
50
60
0
2
4
6
8
10
12
14
16


10000
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
Z
0
5
10
15
20
25
30
35
40
45
50
$\hat{w}$
E_c
h
0.5
4300
4000
0.8
3000
0.2
2000
1000