IMPROVED STORAGE CAPACITY IN CORRELATION MATRIX MEMORIES STORING FIXED WEIGHT CODES

Stephen Hobson and Jim Austin

Department of Computer Science,
University of York, York, UK

Abstract

In this paper we introduce an improved binary correlation matrix memory (CMM) with better storage capacity when storing sparse fixed weight codes generated with the algorithm of Baum et al. [10]. We outline associative memory, and describe the binary correlation matrix memory— a specific example of a distributed associative memory. The importance of the representation used in a CMM for input and output codes is discussed, with specific regard to sparse fixed weight codes. We present an analysis of the benefits of an algorithm for the generation of fixed weight codes, originally given by Baum et al. [4]. The properties of this algorithm are briefly discussed, including possible thresholding functions which could be used when storing these codes in a CMM; L-max and L-wta. Finally, results generated from a series of simulations are used to demonstrate that the use of L-wta as a thresholding function provides an increase in storage capacity of around 15% over L-max.

1 Introduction

Traditional computer memories store data in a compartmentalised fashion, with each stored item having a unique address. While this leads to perfect recall in all cases where the correct address is known, any amount of error in the address will result in an incorrect recall. A contrasting model of memory is *distributed associative memory*. In such a model, data items are stored as pairs, with the presentation of the first member of the pair to the memory resulting in the recall of the second. Rather than these associations being stored in a single location in the memory, they are distributed across the memory. This provides robustness in the presence of noise on the input, and enables generalisation [3].

Such a memory serves a different purpose to a traditional memory. While a standard computer memory is well suited to tasks such as storing a list of tasks or events, it is less capable of "answering questions" [8]. Such a task would require the question to be looked up in a list, which might contain the location of the answer. In a distributed associative memory, the answer is retrieved simply by presenting the question to the input. The recall operation does not require a look-up algorithm, and so is a much more efficient operation.

A Binary Correlation Matrix Memory (CMM) [10] is one example of a distributed associative memory. Its storage capacity is dependent on the data which is stored in it. More specifically, if a set of codes exhibits high orthogonality, the noise on the output will be low. The fixed weight code generation algorithm of Baum et al. [4] has the benefit of generating unique codes which are well separated in the pattern space, which makes them suited for storage in a CMM. We will subsequently term the codes generated from this algorithm *Baum codes*, for ease of reference. Recall of such codes from a CMM requires the use of a thresholding function, and it is with the nature of this function that this paper is concerned. L-max thresholding [1] has been shown to be an effective thresholding function for fixed weight codes, and has been applied to CMMs storing Baum codes [2]. However, L-max thresholding fails to make use of all the constraints on Baum codes. A thresholding mechanism which takes advantage of these constraints is able to provide an improved storage capacity for a CMM which stores Baum codes. We will introduce such a thresholding function, named L-wta, and demonstrate that it is effective through simulations.

2 Binary Correlation Matrix Memories

A Binary Correlation Matrix Memory (CMM) stores the correlations between input and output codes. The memory is a one layer fully connected neural network. This means that the weights can be viewed as an $m \times n$ matrix W, where m is the size of the input vectors and n is the size of the output vectors. An example of such a memory is shown in Figure 1. Although it is possible to use a CMM with non-binary weights [7], only the binary case will be considered in this paper.

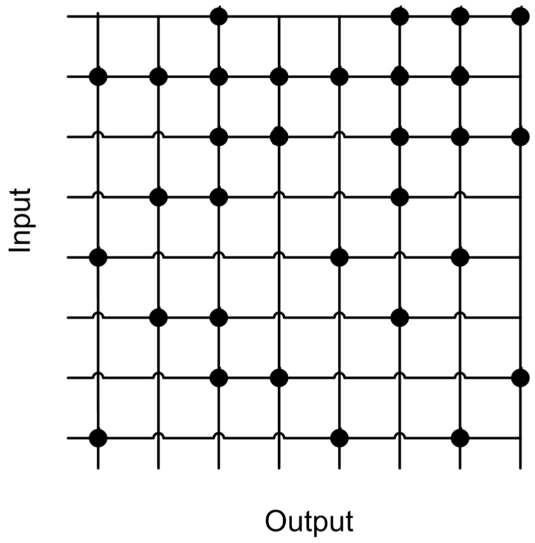

Figure 1. An example of a CMM with 8 input neurons, 8 output neurons and binary weights.

Learning is achieved using an outer product of the input and output. These matrices are combined using an OR function over all input output pairs to create the weight matrix W.

$$W = \bigvee_{i=1}^{N} x_i y_i^T \tag{1}$$

Recall is achieved as shown in Equation. 2.

$$y = f[Wx] \tag{2}$$

Here f is a thresholding function, which takes the activity output Wx and converts it to a binary vector. For example, Willshaw et al. [10] suggested that the thresholding function could set all output nodes with activity greater than the number of 1s in the input pattern x to 1. The choice of this threshold function has a profound effect on the storage capability of the network, as we shall see later.

When recalling a pattern from the memory, the resulting vector (before thresholding) can be viewed as a signal (the original stored pattern) and some noise (extra activity from overlaps with other learned codes). This is demonstrated in Equation 3. [6]

$$y = y_i + \sum_{j=1, j \neq i}^{m} \cos(x_j, x_i) y_j \tag{3}$$

The term y_i is the stored vector we are trying to recall (the signal), whilst the other term is the noise. The magnitude of the noise is dependent on the similarity between between the input vectors x_j and x_i. If the set of input vectors is an orthonormal set, there will be no noise, and hence perfect recall. Of course, using an orthonormal set requires that we either have a small number of input values or a huge number of input neurons, and so is not usually practical. However, if we seek to maximise storage capacity it is beneficial to maximise the orthogonality between the input codes.

3 Sparse Fixed Weight Coding

As we have just seen, in a neural memory such as a CMM there is an intrinsic link between the data representation used and the storage capability of the memory. The encodings chosen for the input and output of the memory will have an effect on the storage capacity. In addition, the choice of threshold function will also relate directly to the representation used, and will affect the ability of the memory to accurately recall stored pairs.

Perhaps the simplest representation of all would be the use of unary output codes (a single bit set to 1 in n bits). This provides a storage capacity of exactly n code pairs. Each input code will be stored in exactly one column of the matrix, and given a correct input code there will be no error on recall. However, the fault tolerance capability of the network is lost, since the storage is no longer distributed. It is necessary to use input and output codes with more than one bit set to 1 to distribute storage over the network.

Furthermore, in order to maximise the storage capability of the network, these codes should be sparsely coded. The number of 1s in an n bit code is termed the *weight* of the code. An important property of the use of sparse codes in a CMM is that the memory is capable of storing $k > n$ codes (where k is the number of pairs stored, and n is the number of input neurons), providing a small amount of recall error is tolerated [9].

If codes with fixed weight are used, an alternative threshold function is available; L-max thresholding [1]. This sets the l neurons with the highest output activity to 1 and the rest to 0, where l is the weight of the output code. For example, consider an output vector with weight 3 and length 8. If the activity of the output neurons is [3 1 3 2 2 0 3 1] then L-max thresholding sets the bits in the positions of the 3 highest values in the vector (in this case the three 3s) to 1, and the other bits to 0, giving an output of [1 0 1 0 0 0 1 0]. Casasent and Telfer [5] experimented with various output encodings, including Binary codes, Hamming codes and fixed weight codes, albeit with analog input codes. They found that in the presence of noise, fixed weight codes with L-max thresholding gave the greatest storage capacity for a given code length.

It is important to have the ability to generate fixed weight codes in such a fashion that the codes generated are guaranteed to be well separated in pattern space, giving a set of codes which maximises orthogonality. Baum et al. proposed an algorithm which generates fixed weight codes which have high orthogonality [4]. The code is divided into l sections which are relatively prime[1] (coprime) in length, with each section i having length p_i. For example, a code of length 32 where $l = 3$ could be divided into sections of length 16, 9 and 7. The size of l defines the weight of the code. To generate code number c, we set the bit in position j as follows (where x is the code to output):

$$\begin{aligned} x_j^c &= 1 \text{ if } j - \sum_{k=1}^{i-1} p_k \equiv c \pmod{p_i} \\ &= 0 \text{ otherwise} \end{aligned} \tag{4}$$

Essentially what is happening is that a single bit will be set to 1 in each section of the code. As subsequent codes are generated, the next bit in each section will be set to 1 instead, wrapping around to the beginning of the section when the end is reached. For example, Figure 2 shows a code with $n = 10$ and $l = 3$, taking $p_1 = 5, p_2 = 3, p_3 = 2$.

$$\begin{array}{ccccc|ccc|cc}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots
\end{array}$$

Figure 2. An example of the generation of Baum codes. Here, $l = 3$ and $p_1 = 5, p_2 = 3, p_3 = 2$, giving a code of length 10

Using this mechanism $p_1 \times p_2 \times \ldots \times p_s$ unique codes can be generated, which is substantially fewer than it is possible to represent with a general fixed weight coding scheme, which is given as $(\frac{n!}{(n-l)!l!})$. However, the overlap between the codes is guaranteed to be small, which improves recall accuracy if they are used in a CMM. Since the method is deterministic, we can be certain about the amount of overlap between generated codes. With no loss of generality we can consider a Baum code with section lengths $p_1 < p_2 < \ldots < p_3$. The first p_1 codes generated will have no overlap at all. The first $p_1 p_2$ overlap by at most 1 bit (a Hamming distance of at least $2l - 2$[2]). In their analysis Baum et al. [4]

[1]Two integers are relatively prime if they have no common factor other than 1. It should be noted that the problem of generating a set of relatively prime numbers which sum to a total is not trivial. However, a discussion of methods is beyond the scope of this paper.

[2]The Hamming distance between two codes is the number of bits which differ between them.

state that if $\prod_{i=1}^{t} p_i$ codes are used, the minimum Hamming distance between any two codes will be $d = 2(l - t + 1)$. For this reason, it is beneficial for $p_i \approx n/l$, since this maximises the product between the section lengths p_i, and hence the number of codes which can be generated with minimal overlap.

4 Improving the Storage Capacity

In the past the algorithm of Baum et al. [4] has been used to generate fixed weight codes, with L-max used as the thresholding function [2]. This represents an oversight, since L-max thresholding may produce output codes which are not possible under the Baum algorithm. By constraining the threshold function so that only the codes generated by the algorithm are output, we show that an increased storage capacity can be achieved.

It has already been mentioned that the algorithm divides the code into a series of sections. Baum et al. point out in the appendix to their paper that a useful property of the algorithm is that there is exactly one 1 in each section of the code. This means that a winner-takes-all (WTA) threshold can be applied to each section of the code, rather than taking the l highest values from the whole code, as we would with L-max [4]. This suggestion was not explored in the paper. We show here the improvement it affords when combined with the L-max method. This thresholding technique incorporates more information about the output encoding into the thresholding function, and therefore provides a more robust thresholding. We shall call this thresholding technique L-wta.

For example, consider an output code of size 8, with $l = 2$. If this code is constructed using Baum codes with section sizes of $p_1 = 5$ and $p_2 = 3$ then this information can be used to improve the quality of recall when compared to L-max thresholding. So, if the output activity is [1 3 4 0 2 1 0 2] then an L-max threshold would give the output vector [0 1 1 0 0 0 0 0]. However, given our knowledge of the method used to construct the code we can be certain that this output is in error, since there must be a single bit set to 1 in the first 5 bits of the code, and a single bit set to 1 in the final 3 bits of the code. In order to apply L-wta to the output code we divide the output activity into groups according to the section sizes used, in this case [1 3 4 0 2] and [1 0 2]. Applying WTA to both sections we get [0 0 1 0 0] and [0 0 1], giving an output code of [0 0 1 0 0 0 0 1].

5 Results

To demonstrate the improved storage capacity of a CMM when using L-wta compared to L-max a series of simulations were conducted. The storage of a CMM is affected by the size of the input and output codes, and also by the weight of the coding system used. For this reason L-wta technique was compared to L-max for a variety of coding systems. In each experiment an empty CMM was created for the appropriate code sizes. The following steps were then undertaken.

1. Generate an input code according to the algorithm of Baum et al. [4] This code will be unique.
2. Select a random example of a Baum code.
3. Train the CMM using the generated input and output pair.
4. Present every input which the CMM has learnt and compare the correct output to the actual output using L-max and L-wta.
5. If average error (defined below) exceeds 10% for all thresholding techniques then exit, otherwise return to 1.

For each experiment these steps were averaged over runs on twenty CMMs. A different integer was used to seed the random generator for each CMM, resulting in differing output patterns being trained. After each iteration the average error was calculated for all twenty CMMs. The recall error was defined as the percentage of recalled patterns which contained an error in any bit. In order to measure the performance of the thresholding techniques at a variety of error tolerances we examine the number of codes learnt in each memory before recall error exceeded 0.1%, 1%, 5% and 10%.

Table 1 shows the results when the size of the input was varied, whilst size and weight of the output code remained constant. Similarly, Table 2

Table 1. Experimental results when varying the size of the input code. All tables show the number of codes learnt before errors at given levels. Codes are given in the format *length/weight*. Note that in some cases code lengths are approximate. This is due to the complexity of generating large sets of coprime numbers which sum to a given target. Bold numbers show the percentage increase in storage capacity when using L-wta rather than L-max.

Input	Output	0.1% error			1% error			5% error			10% error		
code	code	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%
64/4	256/4	70	78	**11.4**	115	129	**12.2**	153	171	**11.8**	179	207	**15.6**
128/4	256/4	139	141	**1.4**	197	234	**18.8**	289	334	**15.6**	345	406	**17.7**
256/4	256/4	259	286	**10.4**	424	473	**11.6**	600	696	**16.0**	719	831	**15.6**
512/4	256/4	496	589	**18.8**	814	927	**13.9**	1182	1369	**15.8**	1416	1630	**15.1**

Table 2. Experimental results when varying the weight of the input code.

Input	Output	0.1% error			1% error			5% error			10% error		
code	code	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%
512/2	256/4	260	260	**0.0**	282	304	**7.8**	482	555	**15.1**	577	707	**22.5**
512/4	256/4	496	589	**18.8**	814	927	**13.9**	1182	1369	**15.8**	1416	1630	**15.1**
512/8	256/4	1023	1036	**1.3**	1453	1603	**10.3**	1811	1989	**9.8**	2028	2229	**9.9**
512/16	256/4	1186	1267	**6.8**	1408	1512	**7.4**	1673	1824	**9.0**	1829	1989	**8.7**

Table 3. Experimental results when varying the size of the output code.

Input	Output	0.1% error			1% error			5% error			10% error		
code	code	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%
256/4	64/4	91	120	**31.9**	178	203	**14.0**	245	286	**16.7**	287	337	**17.4**
256/4	128/4	148	179	**20.9**	265	289	**9.1**	362	429	**18.5**	436	508	**16.5**
256/4	256/4	259	286	**10.4**	424	473	**11.6**	600	696	**16.0**	719	831	**15.6**
256/4	512/4	373	415	**11.3**	608	712	**17.1**	950	1099	**15.7**	1137	1327	**16.7**

Table 4. Experimental results when varying the weight of the output code.

Input	Output	0.1% error			1% error			5% error			10% error		
code	code	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%	L-max	L-wta	%
256/4	512/2	564	709	**25.7**	1259	1435	**14.0**	1932	2138	**10.7**	2310	2599	**12.5**
256/4	512/4	373	415	**11.3**	608	712	**17.1**	950	1099	**15.7**	1137	1327	**16.7**
256/4	512/8	257	267	**3.9**	353	400	**13.3**	495	569	**14.9**	580	677	**16.7**
256/4	512/16	71	89	**25.4**	138	157	**13.8**	197	222	**12.7**	219	266	**21.5**

shows the results when the weight of the input code was varied. In both cases it can be seen that the use of L-wta results in an increase of approximately 15% in storage capacity. L-wta appears to provide the largest increase in storage over L-max when the input code is sparse; that is, when the code size is increased or the weight is decreased. This advantage appears less pronounced as the amount of output error increases.

The case is similar when examining Tables 3 and 4, which show that the effect of output sparsity on the effectiveness of the technique is less clear. However, the storage capacity when using L-wta is consistently a considerable improvement over that achieved using L-max.

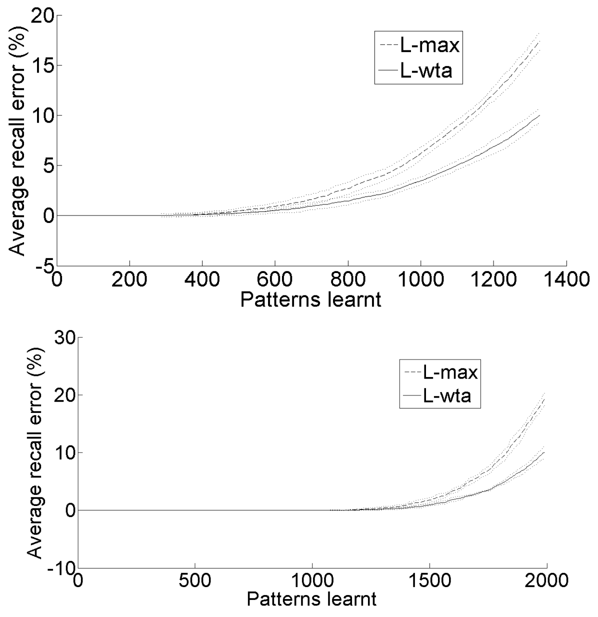

Figure 3. Two comparisons of the storage capabilities of a CMM when using L-max and L-wta. Dotted lines show the standard deviation of average recall error between runs of the experiment. (top) Input codes had size 256 and weight 4. Output codes had size 512 and weight 4. (bottom) Input codes had size 512 and weight 16. Output codes had size 256 and weight 4.

The graph in Figure 3 shows two examples of the performance of the two thresholding techniques as codes are trained into the memories. It can clearly be seen that as the memory becomes increasingly saturated (a large proportion of the bits in the memory are set to 1), the use of L-wta provides an increasing benefit over L-max thresholding.

6 Summary

In summary, it has been demonstrated in this paper that when using codes generated by the algorithm of Baum et al. [4] L-wta provides an increase in storage capacity over thresholding using L-max, provided some error is tolerated. This increase in storage capacity is generally in the order of 15%, but has been observed to be as high as 30%.

Acknowledgements

The authors would like to acknowledge the use of the White Rose Grid (http://www.wrg.york.ac.uk/) in conducting the simulations presented in this paper.

References

[1] J. Austin, T. Stonham, *Distributed associative memory for use in scene analysis,* Image and Vision Computing 5, 251–260, 1987.

[2] J. Austin, J. V. Kennedy and K. Lees, *A neural architecture for fast rule matching,* In Artificial Neural Networks and Expert Systems Conference, 1995.

[3] J. Austin, *Associative Memories,* In Handbook of Neural Computation, E. Fiesler, R. Beale (editors), Oxford University Press, 1997.

[4] E. B. Baum, J. Moody and F. Wilczek, *Internal representations for associative memory,* Biological Cybernetics, 59(4):217–228, 1988.

[5] D. Casasent, B. Telfer, *High capacity pattern recognition associative processors,* Neural Networks, 5, 687–698, 1992.

[6] S. Haykin, *Neural Networks,* A Comprehensive Foundation Second Edition, Prentice Hall, 1999

[7] J. Nadal, G. Toulouse, *Information storage in sparsely coded memory nets,* Network, 1, 61–74, 1990.

[8] G. Palm, *Neural Assemblies,* Springer-Verlag, 1982.

[9] G. Palm, F. Schwenker, F.T. Sommer and A. Strey, *Neural associative memories Associative processing and processors*, 307–326, 1997.

[10] D. J. Willshaw, O. P. Buneman and H. C. Longuet-Higgins, *Non-holographic associative memory,* Nature, 222(7):960–962, 1969.