Fast Associative Memory

Ricardo Miguel Matos Vieira
Instituto Superior Técnico

ricardo.vieira@tagus.ist.utl.pt

ABSTRACT

The associative memory concept presents important advantages over the more common random access memories. It allows the possibility to correct faults and complete missing information in a given input pattern. One of the most important models that represent this concept is the *Lernmatrix*.

The objective of this project is, through the implementation of an associative memory and methods described in this paper, to promote its potential through the resolution of some of its problems. The most pressing issues relate to the inefficiency of this model when implemented over a serial computer and its ineffectiveness when used with non sparse data.

Keywords: Lernmatrix, Associative Memory, Neural Networks, Hopfield Networks, BAM, SDM, Hierarchical Query.

1. INTRODUCTION

A brief description of the concept of associative memory would present it as a content-addressable memory (1). These types of memories work in a different way from traditional memories. The major difference resides in the fact that items are retrieved using its content, rather than a random address (2); in a way, similar to what happens with the human brain. The associative memory is a model that has the objective to, given an input pattern, determine the most similar patterns. Besides this objective, there are several other advantages (3) (4):

- It is possible to correct faults if some information is invalid in the input pattern;
- The ability to complete some missing information in the given vector;

The motivation behind this work is that, although the associative memory model (*Lernmatrix* (5)) presents many advantages over random access memories, it also presents some problems. First, it demonstrates serious performance issues when implemented over a common serial computer. Depending on the data set and hardware available, it may take several hours to execute a single query. The first objective of this project is to improve the query response time. This will be accomplished by reducing the execution time in the event of a *miss* (no matching patterns found). Second, it cannot be applied to information that is not sparse (6), which means that, for instance, it is not possible to store and retrieve a group of images from an associative memory, as a normal binary image has usually half its content filled with black. As soon as more than the two digit number threshold of images is stored in the memory, recalls will always present incorrect patterns.

Regarding the first objective, the proposed solution consists of executing a query on the associative memory in a step by step manner. The solution presented in this work will consist in the aggregation of the data, both in the associative memory and input vector, in order to allow the execution to be faster. This will add execution time if the result is a *hit*, but on a *miss* it is possible to save time. As an example, consider a quadratic associative memory with a size of 2000 units. For an aggregation coefficient of 2 the aggregation process will allow for a query on a 1000 sized associative memory

For the second objective, the proposed solution consists of a method to achieve ideal conditions

for the application of images in an associative memory. As was stated previously, the major setback of using binary images is its ratio on the number of one's it usually presents. The idea is to increase the memory dimension in order to maintain the same information, but achieve the best ratio; for instance, an image with a size of 400, would be represented as a 10000 vector, and would consequently be used in a quadratic 10000x10000 associative memory. This method will increase the threshold of images that can be introduced in the memory before erroneous results start to appear.

2. CAPACITY

Another important topic when referring to Steinbuch's associative memory is its capacity (6) (7). Capacity, when related to neural associative memories, is the maximum number of associations that can be stored and recalled before errors start to appear. As an overview it can be said that, assuming that both vectors have the same size *n* and that both vectors contain *M* ones it has been shown that the most efficient value for *M* is: **M = $\log_2$ (n/4)** and that the *Lernmatrix* capacity is given by **L = ln(2)(n^2/M^2)** (6). *L* is the number of vector pairs that can be stored in this model before mistakes start to occur in the retrieval phase. The capacities for other associative memory models are as follows:

- **Bidirectional Associative Memory** model capacity is 0.199n, where *n* is the binary vector's size (8) (9).
- **Hopfield Network** model capacity is 0.15n (6).

Consider a vector with a size of 10000 units. The optimal value for *M* would be:

$$M = \log_2\left(\frac{10000}{4}\right) \leftrightarrow M = 11$$

For *n* = 10000 and *M* = 11, the storage capacity is:

$$L = \ln(2)\left(\frac{100000000}{121}\right) \leftrightarrow L = 572848$$

The capacities of the other models are:

- **BAM**: $0.199 \times 10000 = 1990$
- **Hopfield's** model: $0.15 \times 10000 = 1500$

The results indicate that in a real case scenario with optimal values, the *Lernmatrix* presents a much bigger capacity than the other models.

3. METHODS

This chapter will explain the mathematics that is behind the solutions to both problems presented in the introduction. It will be divided in the following sections:

Hierarchical associative memory methods: this section will present the formulations for the aggregation and hierarchy of an associative memory with a threshold rule.

Efficient sparse code methods: this section will illustrate the method that permits to code the memory into a higher dimensional space and consequently allow for a better sparseness value.

3.1 Hierarchical associative memory methods

Consider a square associative memory with dimension *n* and *M* number of ones. Assume also that the memory will be implemented in pointer representation. Pointer format is a representation used by many applications that deal with neural networks. Its major selling point is that due to the fact that as data in an associative memory is generally sparse, the zeros in the binary vector can be omitted, saving space and processing time (10); consequently the unit needs only to perform *M* operations instead of *n*. In order to calculate the output of an associative memory *n***M* steps are required.

Consider that an OR aggregation step is introduced, where *a* corresponds to the size of the aggregation. The following formula presents the number of steps required to perform a computation:

$$\frac{M}{a \times n} + M \times a \times M$$

In order to create an aggregation matrix it is necessary M/a operations to compute *n* units. Usually *M* units will have an output of one, but if an aggregation step occurs then instead of *M* we will get *M*a* possible outputs.

The optimal value for *a* is given by:

$$a = \frac{\sqrt{n}}{\sqrt{M}}$$

Consider now a hierarchy of two aggregation steps; *a* corresponds to the size of the first aggregation and *b* the second. The number of steps required to perform a computation will be:

$$\frac{M}{a \times n} + \frac{M}{b \times a \times M} + M \times b \times M$$

The best value for *a* and *b* is:

$$a = \frac{n^{\frac{1}{3}}}{M^{\frac{1}{3}}}, \qquad b = \frac{n^{\frac{2}{3}}}{M^{\frac{2}{3}}}$$

Adding another aggregation *c* will result in the following computation steps:

$$\frac{M}{a \times n} + \frac{M}{b \times a \times M} + \frac{M}{c \times b \times M} + M \times c \times M$$

The optimal values for *a b* and *c* are:

$$a = \frac{n^{\frac{1}{4}}}{M^{\frac{1}{4}}}, \qquad b = \frac{n^{\frac{2}{4}}}{M^{\frac{2}{4}}}, \qquad c = \frac{n^{\frac{3}{4}}}{M^{\frac{3}{4}}}$$

Subsequent aggregations would be calculated the same way as described above. It is important to decide what is the best hierarchy depth (number of aggregation steps). It is supposed that the optimal number of aggregations is given by the following formula:

$$r = \log(n) - \log(M)$$

This is the mathematical basis that was used to calculate the optimal number for each size of aggregation and the hierarchical depth of the three step aggregation process presented in the results section.

3.2 Efficient sparse code methods

There are several methods that allow the mapping of information from an associative memory in another higher dimensional space. The problem that arises from most of these methods is that they use only a small portion of the space of the high dimensional memory. This results in the formation of clusters of data. The method described below does not present this problem.

Consider a binary vector of size *n*. The first step is to divide the vector in sub-vectors according to a window of size *l*. The result of this breakdown will be n/l binary sub-vectors. After this separation the data is converted into unary representation.

Contemplate the following binary vector: 0101011011101101.

For a window of size two, the vector is divided in the sub-vectors:

- 01010110;
- 11101101;

Using a unary representation both sub-vectors will be mapped between 2^0 and $2^l = 2^8$. In this example the first vector would be:

$$2^6 + 2^4 + 2^2 + 2^1 = 86$$

And the second:

$$2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^0 = 237$$

These will be mapped again in binary vectors of size 2^8-1 with each respective weight set to one, and the rest set to zero. In the corresponding example the first vector will have the 86th weight set to one and the rest will be filled with zeros.

As such it can be concluded that the size of the higher dimensional quadratic memory is:

$$2^l \times \frac{n}{l}$$

These were the methods used in the experiments related to the high dimensional associative memory.

4. RESULTS

The previous section demonstrated the mathematics behind the solutions to the problems of efficiency and presenting non sparse feature vectors to Steinbuch's model. The subsequent sub sections will present the experiments and correspondent results.

This section will be divided in the following sub sections:

Hierarchical associative memory: this section presents the results regarding the experimentation of the method described in section 3.1. It relates to the inefficiency problem of the *Lernmatrix* when implemented over a serial computer.

Efficient sparse code: the experiments related to the ineffectiveness of using non sparse data in the *Lernmatrix* are presented in this section. It is used the efficient sparse code method indicated in section 3.2.

4.1 Hierarchical associative memory

Different aggregations were tested and with different numbers of binary vectors stored in the associative memory. All the data is randomly generated. This section is divided in the following subsections:

Test results (no correlation): this sub section presents the experiments with patterns and a memory with a length of 1000 units. The number of ones in each vector is according to the optimal value formula presented in section 3. In this case, in average, each vector has about 7 to 8 ones distributed across its length. No correlation indicates that the patterns are stored in the memory in a sequential operation. The first stored vector filling the first column of the memory and so on. This will result in a sparser memory, populated with less ones.

Test results (with correlation): maintaining the same vector size and number of stored images, this section is distinguished from the previous one by the usage of correlation. Storage correlation indicates that the stored pattern is correlated with itself in order to fill the memory, similar to what happens in Figure 4.

Test results (with correlation and smaller data set): Due to the results in the previous sub section, a smaller data set was produced. The size and number of ones in each vector is the same.

4.1.1 Test results (no correlation)

The first set of experiments was made with an associative memory without storage correlation. The absence of correlation will create a sparser memory. The number of ones populating the memory will be less than with correlation.

In every test, the size of the binary vectors in the associative memory is 1000 elements. The experiments were made by attempting to recall from the memory patterns that were not previously stored.

Aggregation coefficient 2

The first experiment was made with an aggregation coefficient of 2. This means that both the input pattern and the associative memory will be reduced from 1000 units to 500 units.

Figure 1 presents the results for both the *normal* and the optimized associative memory. The *Y* axis represents the time in *ms*, and *X* relates to the number of binary vectors previously stored in the memory. As can be seen there is a *speed up* when the query is executed in the aggregated associative memory before the *normal* one.

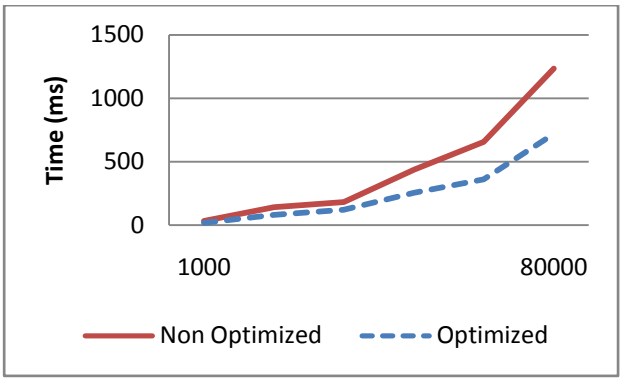


Fig. 1 Execution Time with Aggregation 2 (no correlation).

Aggregation coefficient 10

As Figure 2 illustrates, a higher coefficient does not imply a major decrease in the execution time.

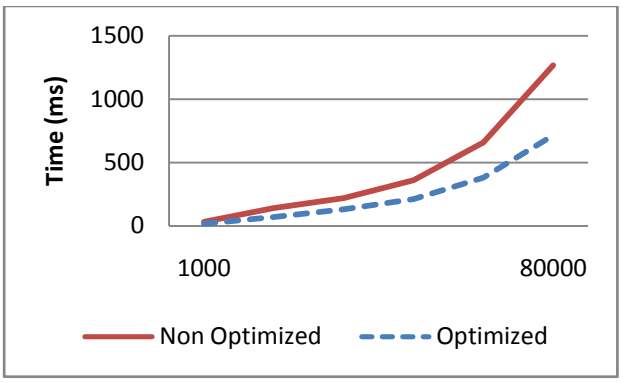


Fig. 2 Execution Time with Aggregation 10 (no correlation).

As this is a time related experiment, there may be some fluctuations related to the way the computer operating system works. Therefore it was also measured the number of operations in all four experiments. As can be seen in Table 1 there is a difference between the number of executed operations in each aggregation. As expected, the higher the aggregation coefficient is, the smaller the number of executed operations.

Three step aggregation (25, 10 and 5)

The objective of combining the aggregations in various steps is to smooth the execution time. In one hand, some input patterns will execute one or two aggregation steps. Others will go through the various steps until a similar pattern is returned in the final step. As can be seen in Figure 3 a three step aggregation process seems to be ideal if compared to the execution of each aggregation step individually. Execution times were remarkably lower than in the previous experiments.

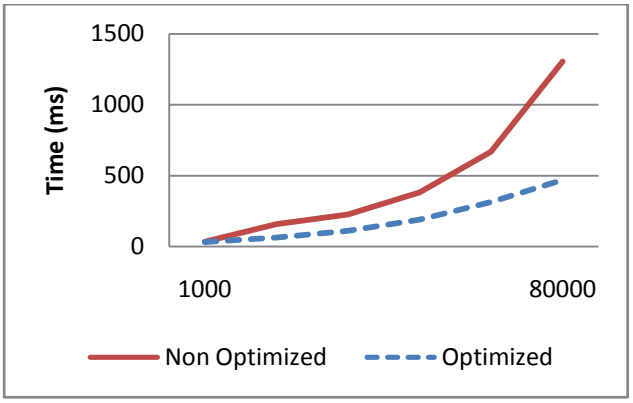


Fig. 3 Execution Time with a three step aggregation process.

The results indicate that there is potential in the application of this method before the query is presented to the *normal* associative memory. As data sets go larger, the bigger the gap between the optimized and non-optimized execution times.

4.1.2 Test results (with correlation)

The second part of the experiments was made with correlation in the associative memory. As it is intended, the number of ones in the memory will be higher than without correlation. The same tests were executed, which means, same number of images and same steps of aggregation.

Aggregation coefficient 2

The results presented in Figure 4 lead to the conclusion that above the 5000, 10000 images, the memory always returns a matched pattern. In this case, the execution time will always be higher in the optimized associative memory, as both the

Nº Images	1000	5000	10000	20000	40000	80000
Non Optimized	112429	558230	1119836	2229912	4451945	8915002
Aggregation 2	97700	484879	972747	1936425	3866646	7742327
Aggregation 5	95802	474711	953251	1897674	3787924	7586830
Aggregation 10	92369	458941	920030	1834699	3661161	7331978

Table 1 Number of operations for each aggregation coefficient

aggregated and non aggregated steps will be performed.

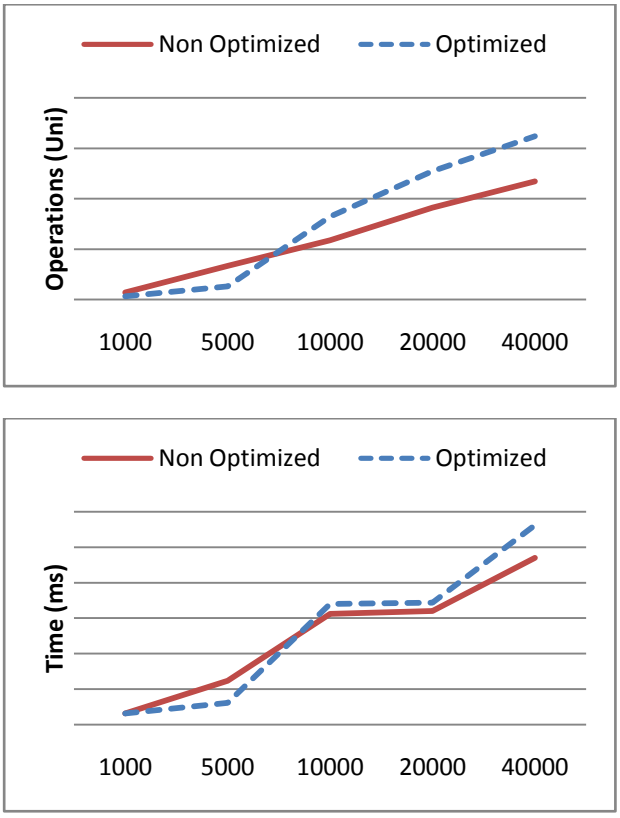


Fig. 4 Execution time and number of operations for Aggregation 2.

It is to be noted that a comparison between the correlated and non correlated experiment, for the 1000 and 5000 size vector tests, indicates that the execution time decreased. This can be explained because the number of ones in the correlated memory is higher than without correlation.

Aggregation coefficient 10

The Aggregation 10 test follows the trend seen in the previous experiments. The probability of finding a matched pattern in a higher aggregated memory is higher, and therefore both the optimized and non optimized memories will be executed.

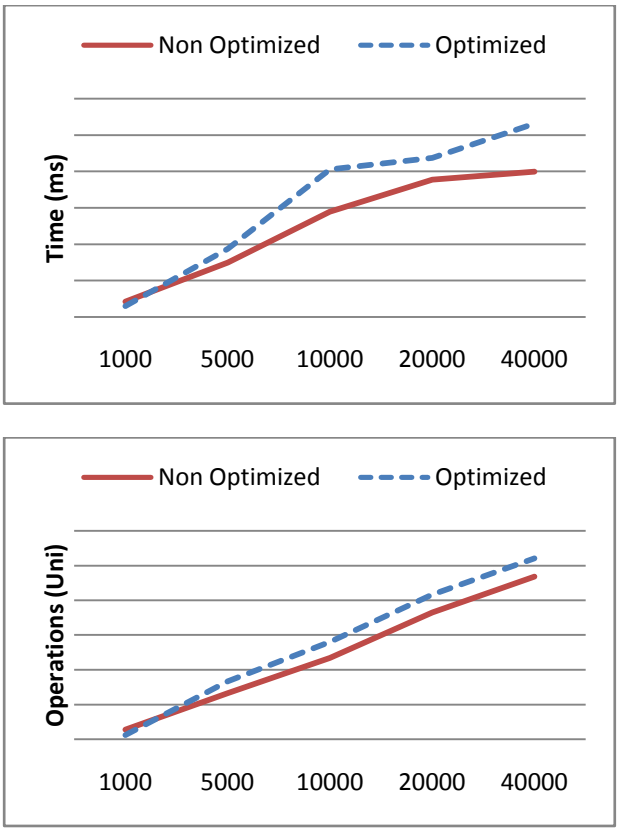


Fig. 5 Execution time and number of operations for Aggregation 10.

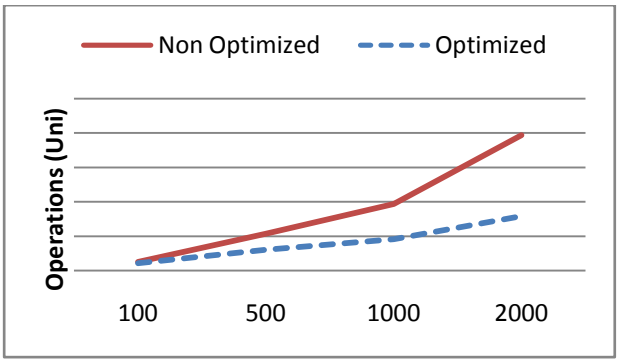
The results lead to the conclusion that for a 1000 size quadratic associative memory the number of binary vectors correlated in the current experiment is too big. The next set of experiments will have reduced data sets.

4.1.3 Test results (with correlation and smaller data sets)

In the following experiments, the number of binary vectors stored in the associative memory will be 100, 500, 1000 and 2000.

Aggregation coefficient 2

As the results below illustrate (see Figure 6), with a smaller number of correlated vectors the probability of finding a matched pattern is lower and with it increases the value of applying the aggregation method.



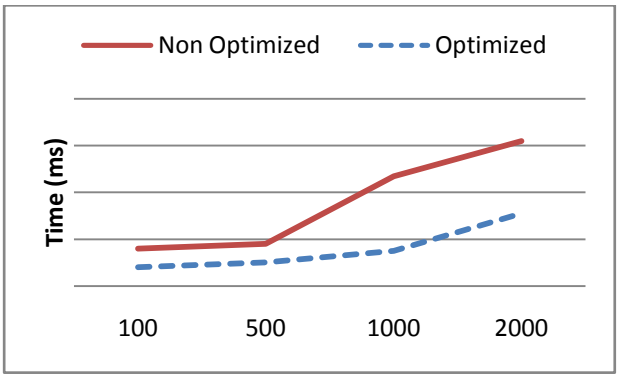
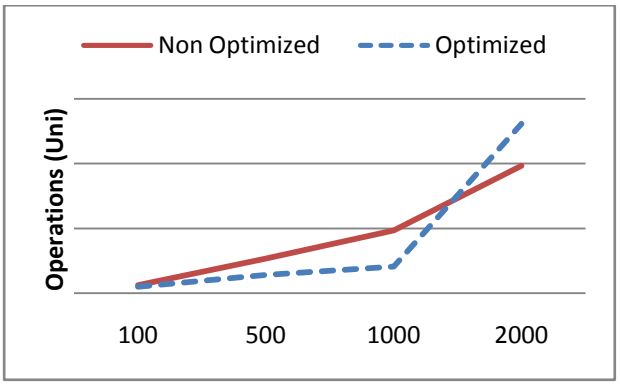


Fig. 6 Execution time and number of operations for Aggregation 2.

Aggregation coefficient 10

The aggregation 10 experiment indicates that, as the memory is shrunk ten times, the probability of finding a matching pattern is already big enough that a 2000 vectors data set will find a match and force the process to execute both the aggregated and non aggregated steps (see Figure 7).



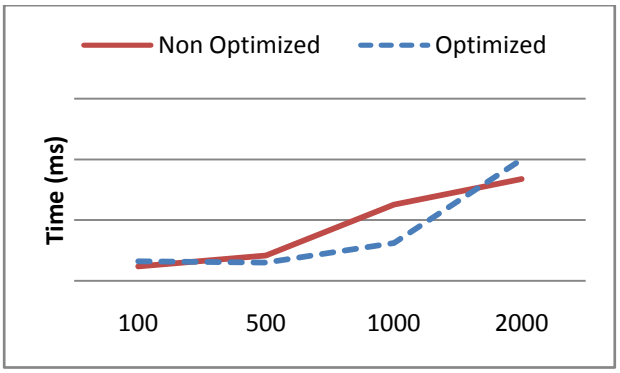
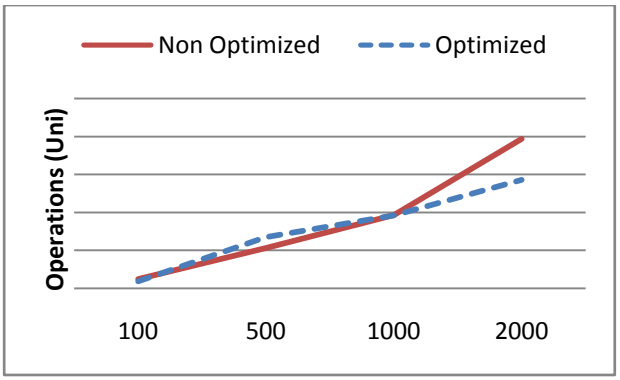


Fig. 7 Execution time and number of operations for Aggregation 10.

Three step aggregation (25, 10 and 5)

The three step aggregation experiment presents mixed results. Due to its nature, it is harder to evaluate how fast the process is. In some data sets, the input may be found in 2 or even 3 steps of aggregation which means that in the end it largely depends on the input pattern.



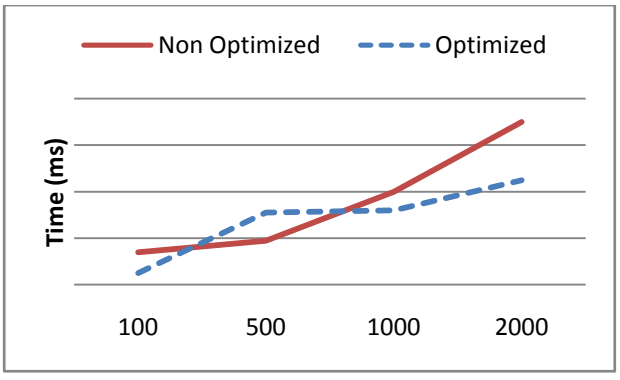


Fig. 8 Execution time and number of operations for three step aggregation method.

As a conclusion of this set of experiments the aggregation method can be evaluated positively. Its usage can reduce the number of operations and

time wasted in trying to find a pattern that will not exist in the *normal* associative memory.

4.2 Efficient sparse code

This section refers to the experiments and results achieved regarding the solution to the problem of using non sparse data with Steinbuch's associative memory. The following tests will be presented throughout this sub section:

- Binary images with 50, 30, 20 and 10 percent of ones;
- For each of the above mentioned data, data sets of 50, 100, 200 and 400 images;

The graphics shown below will plot the number of errors (Y axis) for each data set (X axis). A recalled pattern is considered an error when the number of ones differs from the inputted image by more than thirty percent.

Fifty percent ones

As can be seen in Figure 9 the number of errors for the query in the *normal* memory is always very high. As the inputs are randomly generated, there are some variances.

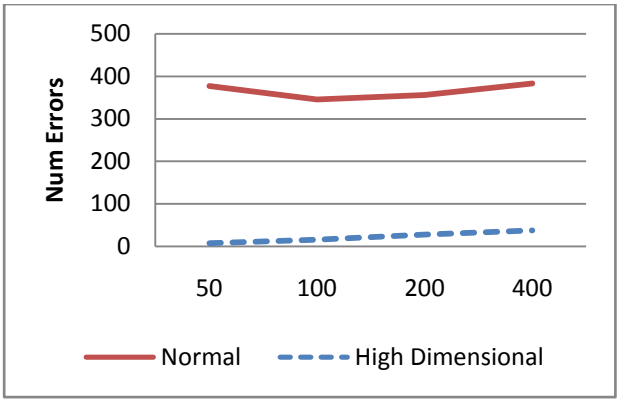


Fig. 9 Number of errors in both the high dimensional and *normal* associative memories for 50 percent ones.

Thirty percent ones

With only 30 percent of ones and a 50 image data set, the *normal* associative memory is still not full (see Figure 10). As was illustrated in Figure 9, the high dimensional space memory presents a lower number of errors that is steadily increasing as the number of stored patterns grows.

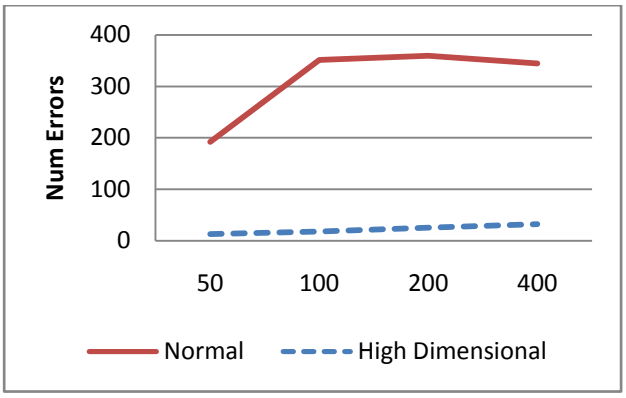


Fig. 10 Number of errors in both the high dimensional and *normal* associative memories for 30 percent ones.

Twenty percent ones

As expected, because the number of ones in the images is decreasing, the associative memory is becoming sparser. Figure 11 indicates that there is a linear correspondence between the number of stored images and the errors on the *normal* memory.

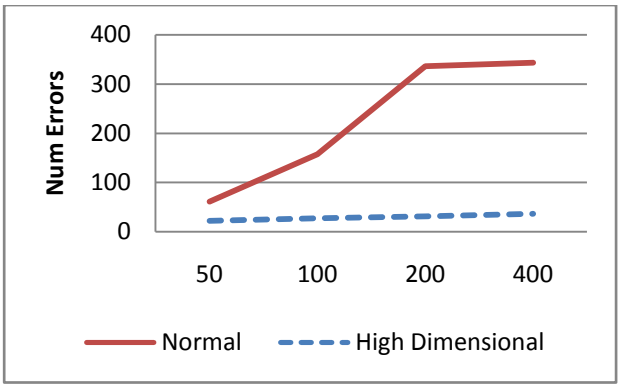


Fig. 11 Number of errors in both the high dimensional and *normal* associative memories for 20 percent ones.

Ten percent ones

The results show that the number of errors for both memories is similar when neither is full. As predicted, when the matrix becomes filled with ones the number of resulting errors increases (see Figure 12).

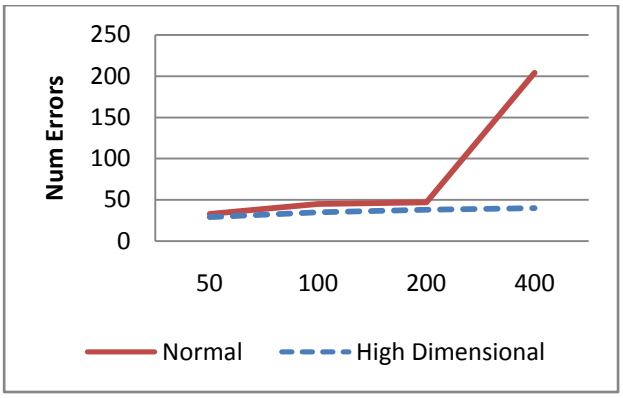


Fig. 12 Number of errors in both the high dimensional and *normal* associative memories for 10 percent ones

It is important to find out when the high dimensional space gets full. In order to discover this, a second group of data sets were created, larger than the previous ones.

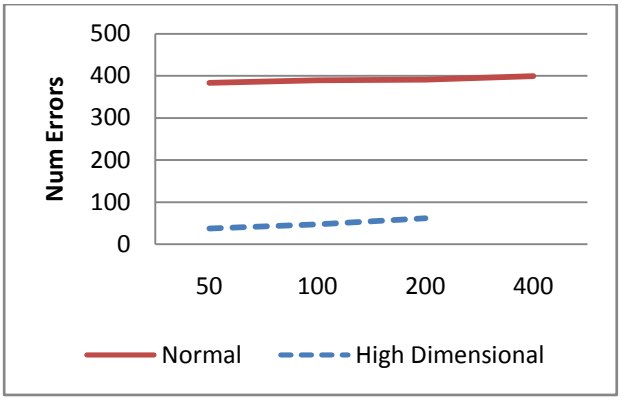


Fig. 13 Number of errors in both the high dimensional and *normal* associative memories for 50 percent ones.

Due to a test plant limitation it was not possible to continue the experiments after the 1200 images threshold. However, the results indicate that even with 1200 images the number of errors is still very low, which demonstrates the potential of this method (see Figure 13).

5. CONCLUSION

The principal objective of this paper was the improvement of Steinbuch's *Lernmatrix* functionality and efficiency. Two particular cases were considered, the first was the efficiency of the associative memory method when a *miss* often occurs and the second the usage of non sparse information. In order to achieve these objectives it was implemented an associative memory with the functionality described by the methods presented in section 3. Through the results illustrated in section 4 it can be concluded that these objectives were fulfilled. Both the hierarchical associative and efficient sparse code methods present potential.

Regarding the hierarchical associative memory method, it can be said that depending on the size of the data set, efficiency can be improved by up to 50 per cent (see Figure 1). This mechanism also presents some disadvantages. It is necessary to compute the aggregated memory before the patterns are presented to the matrix in the recall phase; consequently this new step will increase the initial computation and storage requirements in the learning phase. It was also concluded that there are limitations regarding the relation between the size and number of stored patterns in the matrix. If the number of images in the memory is very high compared with the size of each image, then the efficiency will be degraded. This relation is deeply connected to the probability of a *hit* or a *miss* in the recall phase.

The coding method that allows the computation of a higher dimensional memory also presents good results. As was demonstrated in the section 4 the number of errors after the recall of a non sparse pattern is several times lower if a high dimensional space is used instead of the *normal* one. Due to some experiment problems it was not possible to find out for which values will this method start returning too many incorrect patterns. Another discovery that was made in the process of these tests is that there is some high dimensional space that is not used. This behavior is justified by the conjunction of the unary to binary translation with the correlation in the storage phase.

As a conclusion it can be said that both these methods present a way to improve the functionality to one of the most important concepts in the field of neural networks, Steinbuch's Associative Memory.

5.1 Future work

The experiments illustrated in this paper represent only the initial step. A lot of work can be done to improve these methods. Regarding the efficient sparse code method, in the mathematical field, it is important to find a way to calculate the optimal window size depending on the sparseness of the data and size. It would also be interesting to optimize the coding mechanism in order to use all the high dimensional space.

Furthermore it is important to remember that all tests were done with random generated binary images. Future experiments should be made to apply these methods in real databases.

There are not many applications of the *Lernmatrix* in office databases majorly because of its steep requirements for the sparseness factor in each input pattern. The high dimensional matrix method is an opportunity to test and potentiate its application.

6. BIBLIOGRAPHY

1. **S. Brunak, B. Lautrup.** *Neural Network Computers with Intuition.* s.l. : World Scientific Publishing Company, 1990.

2. **Kohonen, Teuvo.** *Self-Organization and Associative Memory.* s.l. : Springer, 1989.

3. **Palm, G.** *Neural Assemblies: an alternative approach to artificial intelligence.* s.l. : Springer-Verlag New York, Inc. , 1982.

4. **Anderson, J.** *Cognitive Psychology and its Implications.* s.l. : Worth Publishers, 1995.

5. **Steinbuch, K.** *Die Lernmatrix (Automat und Mensch).* s.l. : Springer-Verlag, 1961.

6. **Hecht-Nielsen, Robert.** *Neurocomputing.* s.l. : Addison-Wesley, 1989.

7. **G.Palm, F. Schwenker, F. Sommer, A. Strey.** *Neural Associative Memories.* s.l. : IEEE Computer Society Press, 1997.

8. **Kosko, B.** *Neural Networks and Fuzzy Systems: A Dynamic Systems Approach to machine intelligence.* s.l. : Prentice Hall, 1991.

9. **Tanaka, T.** *Capacity Analysis of BAM.* s.l. : J. Phys. Soc. Jpn. 73 pp. 2406-2412 , 2004.

10. **H. Bentz, M. Hagstroem, G. Palm.** *Information storage and effective data retrieval in sparse matrices.* s.l. : Elsevier Science Ltd., 1989.