INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Fast Associative Memory

Ricardo Miguel Matos Vieira

Dissertation to obtain the Degree of Master in
Computer Engineering

Júri

Presidente: Prof.ª Dra. Ana Maria Severino de Almeida e Paiva
Orientador: Prof. Dr. Andreas Miroslaus Wichert
Co-orientador: Prof. Dr. Pável Pereira Calado
Vogal: Prof. Dr. Jan Gunnar Cederquist

January 2009

Acknowledgments

First of all I would like to thank my adviser Professor Andreas Wichert which helped me way beyond the line of duty. This thesis represents a subject that is dear to his heart and more than a professor and student relationship ours has been one of friends and partners trying to solve a mystery.

I am also grateful to everyone who helped me to uplift the state of the art in the neural networks subject, especially Professor Frantisek Zboril from the Faculty of Information Technology (Czech Republic) who was so kind to send his research in the different associative memory models.

I would also want to thank my family for their endless support, help and suggestion for me to get this project *right* rather than *fast*. My friends who had the patience to listen to my endless bragging whenever things didn't go the way it was expected.

Last but not least I want to thank everyone that I forgot to mention and that contributed in some way to this work. Thank you all.

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. In addition, it also represents the conceptual advantage of being important as its study relates to the functional behavior of the human mind.

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 thesis, 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.

This thesis will describe the research on this subject, beginning with the theoretical framework and state of the art that will contextualize the work. It will be presented some of the most important associative memory models and applications. This will be followed by the reasoning behind the choice to work with the *Lernmatrix* model and the mathematical framework that supports the proposed solutions. Finally, the solutions will be tested and evaluated and summarized in the conclusions chapter.

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

Resumo

O conceito de memória associativa apresenta importantes vantagens sobre as mais comuns memórias de acesso aleatório. Entre outras, permite a possibilidade de corrigir falhas e completar informação que esteja ausente em determinado padrão de entrada. Além destas, apresenta também a vantagem do seu estudo se encontrar fortemente ligado ao comportamento funcional do cérebro humano.

Um dos mais importantes modelos relacionados com este conceito é a *Lernmatrix*. O objectivo desta dissertação é, através da implementação de uma memória associativa e métodos descritos em subsequentes capítulos, promover o seu potencial através da resolução de alguns dos seus problemas. As questões mais prementes prendem-se com a ineficiência deste modelo (*Lernmatrix*) quando implementado sobre um computador de processamento serial e a sua ineficácia na utilização de dados não esparsos.

Esta dissertação irá descrever o trabalho desenvolvido sobre este tema, começando com o enquandramento teórico e estado da arte que irão contextualizar o trabalho. Serão apresentados alguns dos modelos e aplicações mais relevantes. Seguir-se-á o raciocinio por trás da escolha de trabalhar com o modelo *Lernmatrix* e a matemática que suporta as soluções propostas. Finalmente, as soluções serão testadas e avaliadas no capitulo de conclusões.

Palavras-chave: *Lernmatrix*, Memória Associativa, Redes Neuronais, Redes de Hopfield, BAM, SDM, Consulta hierárquica.

Section I

Dissertation

Contents

Figure Index

Table Index

Glossary of terms and acronyms

BAM Bidirectional Associative Memory

SDM Sparse Distributed Memory

Chapter I

Introduction

This project addresses the concept of neural networks in general and more particularly associative neural memories. Artificial neural networks are, simply, models that intend to emulate the structure and functionality of the human brain (1) (2). The initial motivation behind the study of these models is to examine and make use of the human problem solving method (3) (4). The associative memory is a particular class of the artificial neural networks that has gained recent interest due to Hopfield's work in the eighties (5). In order to mirror the way the human organs sense the world, the associative memory models uses vectorial representation (6) (7). Each pattern of information is represented by a vectorial structure that allows for the computation of similarity between them (8). This thesis assumes that the reader has some knowledge in this field of expertise, and therefore will not describe the neural networks concept in detail.

1.1 Motivation

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 (9) (10):

- The possibility 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* (11) (12)), present many advantages over random access memories, it also presents some problems. First, there are performance issues when implemented over a common serial computer. Second, it cannot be applied to information that is not sparse (11), 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.

Consider Figure 1 that represents a common binary image. After its storage in a blank associative memory the result is the matrix diagram depicted below.

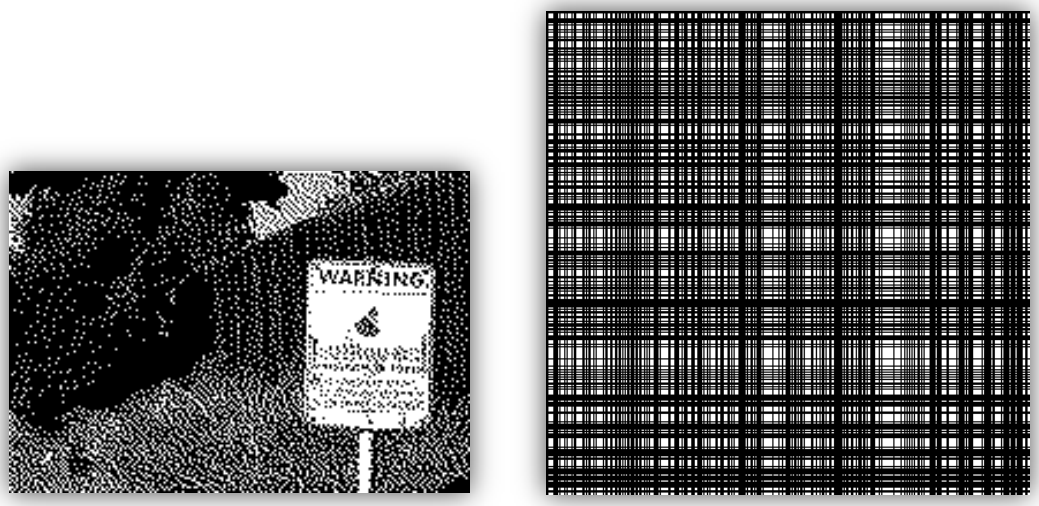


Fig. 1 Binary Image roughly half black and the associative memory after its storage.

Consider that the presence of a one in the matrix is represented by a white pixel, and a zero black. It is not difficult to imagine that after storing some images in the memory it will quickly become full.

Another problem previously referred is the time it takes for a modern day serial computer to resolve queries presented to the *Lernmatrix* (11). The graphic illustrated in Figure 2 represents the average execution time for a single query on a set of associative memories implemented over a serial computer.

- The Y axis represents the query response time;
- The X axis refers to the number of images stored in the memory;

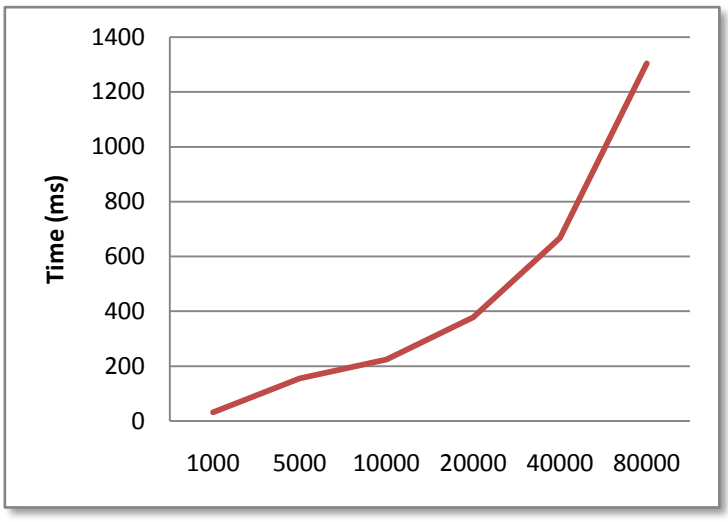


Fig. 2 Graphic representing the execution time of a query on a set of associative memories.

As can be seen, there is a relation between the time and number of stored images. As the number of images and size increases, so does the time it takes to execute a single query.

1.2 Objectives

This thesis deals with the two major problems that were identified in the implementation of the *Lernmatrix* (11) (12). The first relates to the quantity of time it takes to process a query over a serial computer when a large associative memory is being used. Depending on the data set and hardware available, it may take several hours to execute a single query. One of the main objectives 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).

The second major objective is to develop a new sparse coding technique that will allow, for instance, image utilization with the *Lernmatrix* (12). In average, a binary image has close to half its content black, the other half white. One of the distinct requirements of the associative memory is regarding to the amount of one's a given input pattern must have (11). This number will always be much smaller than the vector size. As such, trying to use an associative memory with randomly generated images will result in many query errors. As soon as more than the two digit number threshold of images is stored in the memory, recalls will always present incorrect patterns.

The expected functionality is the minimum needed to demonstrate the achievement of the proposed objectives. It will allow the execution of queries in an associative memory with the implementation of the solutions that will be described in chapter IV.

Later in chapter V and VI, the test results will be presented in the form of tables and graphs. These will represent execution times, number of operations and query errors.

1.3 Solution

Regarding the first objective, the proposed solution consists of executing a query on the associative memory in a step by step manner.

Consider a query which results in a *miss*. The usual associative memory behavior would be to search the whole memory. The solution presented in this thesis 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 in the event of a *hit*, but on a *miss* it is possible to save time. The aggregation method will be explained in chapter IV.

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 unit associative memory. If the result is a *miss*, there is a probability of skipping the second step, which means that it would only be needed half the number of operations. If it is a *hit*, then it will have to be executed in the normal sized memory, adding the initial step execution time as overhead.

For the second objective, the proposed solution consists of a method to achieve ideal conditions for the utilization of images with the *Lernmatrix* (12). 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 optimal 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 will increase the threshold of images that can be introduced in the memory before erroneous results start to appear.

In future work related to this field, it would be important to test the proposed solutions in real databases. At the current level of technology it is improbable that it is possible to test these solutions with large images. Databases with small binary images would be ideal.

The application of this work would be to enable the usage of images in databases using all the advantages of an associative memory. It is important to state that the methods described in this paper can be used with any non sparse patterns, not only images.

1.4 Dissertation Structure

The thesis will be divided in the following chapters:

- **1st** Chapter – **Introduction**: Briefly presents the motivation behind the project, its objectives and solutions.
- **2nd** Chapter – **Related Work**: A rather complete study was made to the current applications of associative memories. This chapter will present some of the work that has been done in this field recently. It will also stress the importance of this project and its potential.
- **3rd** Chapter – **Associative Memory capacity**: This chapter will present the reasoning behind the work on the *Lernmatrix*; what are its advantages in comparison with other models.
- **4th** Chapter – **Methods**: This chapter will describe, rather thoroughly, the solution and mathematical framework that supports it.
- **5th** Chapter – **Hierarchical associative memory results**: This chapter will present the results that occurred from the experiments on the method of hierarchical associative memory.
- **6th** Chapter – **Efficient Sparse code results**: The results chapter will illustrate, in the form of graphs and tables, the results that occurred from the experiments on the associative memory for the efficient sparse code method.
- **7th** Chapter – **Conclusion**: The conclusion will stress the most important knowledge obtained throughout the experiments.

Chapter II

Overview – Related Work

2.1 Associative Memory

A brief description of the concept of associative memory would present it as a content-addressable memory (13). These types of memories work in a different way from traditional memories. The major difference resides in the fact that the items are retrieved using its content rather than a random address (14); in a way, similar to what happens with the human memory.

The human mind correlates information based on the memories it contains. This allows for the recall of a particular event based on the hearing of a small tune, or sniffing an odor you haven't smelled in years. Associative memories have similar behaviors. One small part of the memory is linked and associated with the rest (15).

This characteristic presents associative memories with important advantages over the more common random memories. Some of the more important are as follows (9) (10):

- The ability to correct faults if false information is given;
- The possibility to complete information if some parts are missing;
- To interpolate information, meaning, if a pattern is not stored, the most similar stored pattern is recalled;

The associative memory subject has earned the attention of several researchers throughout the last five decades. One of the first of these researchers was Karl Steinbuch, famous for his work in the *Lernmatrix* (12). The *Lernmatrix* is one of the most important and crucial precedents in the development of current models and after several decades since its release, it still holds true.

An associative memory has a fundamental purpose: to recover the most similar set of patterns, given an input. These patterns are usually represented by binary vectors. An aspect of the pattern that is present in the vector is represented by a one component. A zero component represents the absence of its correspondent aspect. This structure allows for the representation of all kinds of information; words, ontologys etc (16) (17) (18) (19).

The *Lernmatrix* functionality is accomplished in two distinct phases:

1. Learning phase (generation of the associative memory)
2. Recovery phase (operation on the associative memory)

In both phases, it can be described as an input/output system (see Figure 3).

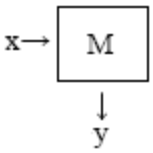


Fig. 3 Diagram representing the *Lernmatrix* input/output system.

The input pattern is represented by a binary vector denoted *x*, and the recall pattern is denoted *y*. *M* represents the associative memory. In the initialization phase, there are no correlations and as such every position in the memory is set to zero.

In the learning phase, pairs of binary vectors are associated. This association is described by the following rule:

$$w_{ij}^{new} = 1 \; if \; y_i \cdot x_j = 1$$

$$w_{ij}^{new} = w_{ij}^{old} \; otherwise$$

This is called the binary Hebb rule (9) (20) (21).

Figure 4 describes the application of the binary Hebb rule on an initialized associative memory.

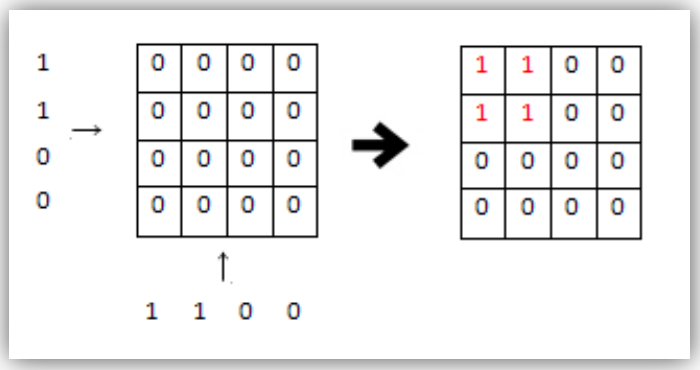


Fig. 4 Binary Hebb rule in the learning phase on an initialized associative memory.

In the retrieval phase, an answer vector *y* is recalled from the associative memory. The recalled pattern depends on the presented question vector *x* and is the most similar vector stored in the memory. The rule that determines the answer vector *y* is as follows:

$$y_i = \begin{cases} 1 \sum_{j=1}^{n} w_{ij} x_j \geq T \\ 0 \; otherwise. \end{cases}$$

T corresponds to the threshold. There are two kinds of strategies to determine thresholds. They are called the soft and hard threshold strategies. As a brief explanation:

- In the hard threshold strategy, *T* is set to a minimum number of *one* components in each unit of the memory. A response may not be found using this strategy.

- The soft threshold strategy sets the threshold as the sum of each unit. As such, a matched pattern will always be recalled.

In the example below, it was a used a hard threshold strategy where *T* equals 1.

Figure 5 describes a simple retrieval, on an already filled associative memory.

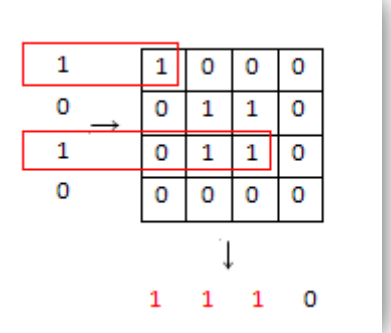


Fig. 5 Retrieval phase on an already filled associative memory.

Another important topic when referring to Steinbuch's associative memory is its capacity (11) (22). Capacity, when related to neural associative memories, is the maximum number of associations that can be stored and recalled before errors start to appear. This subject will be described in detail in the 3rd chapter. 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: $\mathbf{M = \log_2 (n/4)}$ and that the *Lernmatrix* capacity is given by $\mathbf{L = \ln(2)(n^2/M^2)}$ (11). *L* is the number of vector pairs that can be stored in this model before mistakes start to occur in the retrieval phase. As will be explained later, the asymptotic storage capacity of Steinbuch's model is far greater than other associative memory models, assuming that the optimal value for *M* is used.

The information above corresponds to the mechanisms and methods used in the *Lernmatrix*. The following sections will present and briefly describe other models related to the concept of associative memory.

2.2 Hopfield Networks

The Hopfield Network is an associative memory model presented by John Joseph Hopfield in 1982 (23). Hopfield's work was a success because he showed that it is possible to solve computational problems using physical systems. These could be, theoretically, implemented in hardware using common and standard components. Nowadays, Hopfield networks are usually applied in the classification of problems with binary pattern vectors (24) (25) (26).

Like all associative memories, the Hopfield network is created by storing input data binary vectors. These vectors, also called *patterns,* represent classes in which every following inputted *pattern* will be mapped. In an n-dimensional data space, the class *patterns* should have *n* binary components. The network is then used to classify the *patterns* that are presented to the memory into the existing classes. Usually, the associated vector will then

belong to one of the class *patterns*, but in a worst case scenario some may be wrongly associated.

There are two types of Hopfield networks, a continuous and a discrete time version. Both follow the dynamics of the formula given below (25).

$$W = \frac{1}{n}\sum_{i=1}^{D} \varepsilon_i^T \varepsilon_i$$

This represents the calculation of the weights *W* for the Hopfield network matrix. In the above formula, *D* represents the number of class patterns $\{\varepsilon_1, \varepsilon_2, ..., \varepsilon_D\}$, that are to be stored in the network; *n* is the dimension of the class *pattern* vectors.

Figure 6 describes a simple recall of a distorted pattern on a Hopfield network[1].

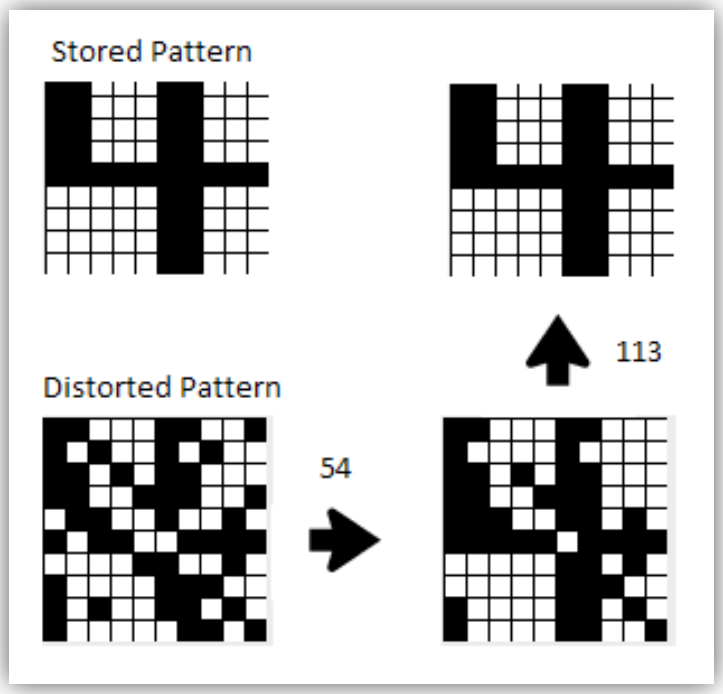


Fig. 6 Iteration process of a Hopfield Network on a distorted pattern.

The numbers near the arrows represent the iterations on the network before reaching each stage. Regarding the capacity of the Hopfield Networks model, it has been calculated to be around 0.15*n*, where *n* refers the number of units (11).

2.3 SDM – Sparse Distributed Memory

The sparse distributed memory is a basic model of an associative memory developed as a mathematical long-term memory by *Pentti Kanerva* in 1988 (27) (15). The discovery of the model was led by the idea that the distances between concepts in our brain, correspond to the distance between points in a high-dimensional space. The basis of the model relies in the fact

[1] **Kriangsiri Malasri,** *Hopfield Network Applet v1.3.* 2001.

that if a concept, an experience, or any kind of information is represented by a high-dimensional vector, then that representation needs not to be exact. Any point of the memory space that might be one of those concepts or experiences is relatively far from most of the space and consequently from other concepts (27). This means that any concept can be represented using a certain degree of inaccuracy before it is confused with other concepts. This property accounts for the *sparseness* characteristic associated to the model.

This model corresponds to the way humans and animals with advanced sensory systems and minds function. The signals interpreted by our brain at two different times are hardly ever similar, and yet the identification of the source of the signal is usually easy to perform (28).

Another important property of the high-dimensional space has to do with the distances between points. If you search for two random concepts in a high-dimensional space odds are that they are far from each other. In the average, they are probably not correlated. The interesting part is to explore the space between those concepts. Between both selected points of interest there will be many concepts that are close to both, in the sense that the amount of space around an intermediate concept that contains both the original points is very small (27). From this it can be concluded that it will be possible to easily find a concept that links the two unrelated initial concepts.

As can be seen, *Kanerva* developed his ideas in an attempt to provide a computational description of structures in the brain. The effort seems to be successful as SDM can be mapped onto physiological structures, something that many alternative associative memories cannot duplicate. Most neural models only duplicate a style of computation and are not intended to model brain functions.

Having the above explanation in mind, it is possible to understand the necessity for the model to compute the similarity between values. This distance can be calculated the same way we would have for a 2 or 3 dimensional space, using Pythagoras theorem. However, for a simple space, the same functionality can be obtained by using the *hamming distance* approximation (29). The *hamming distance* between two binary values is measured by counting the number of bits which are different.

An explanation is in order to justify the *distributed* property of the model. When a value is written, it is also written into every location in the memory about that location. Likewise, when a value is read, it is calculated by examining the contents of every location in the space about that location. The best match will then be calculated by averaging each bit individually. As can be seen, each address is involved in the storage of many values, and many addresses are involved in the storage of any one value. This is why the memory is *distributed*, because there is no need for the locations to be physically adjacent.

Figure 7 describes an example of writing and reading from a sparse distributed memory with a 256-bit address and data size. The six *patterns* at the top were stored as a sequence. The *patterns* at the bottom resulted from a search in the memory. The initial inputted *pattern* is a corrupted version of the third *pattern*. As can be seen the result is a clear version of the sixth *pattern*.

This example demonstrates how the SDM model is tolerant of errors in the address. Another property that can be verified, and that is not shared by all associative memories, is that the SDM is not only tolerant of corruption in the address but it is also tolerant of noise in the data as well (15).

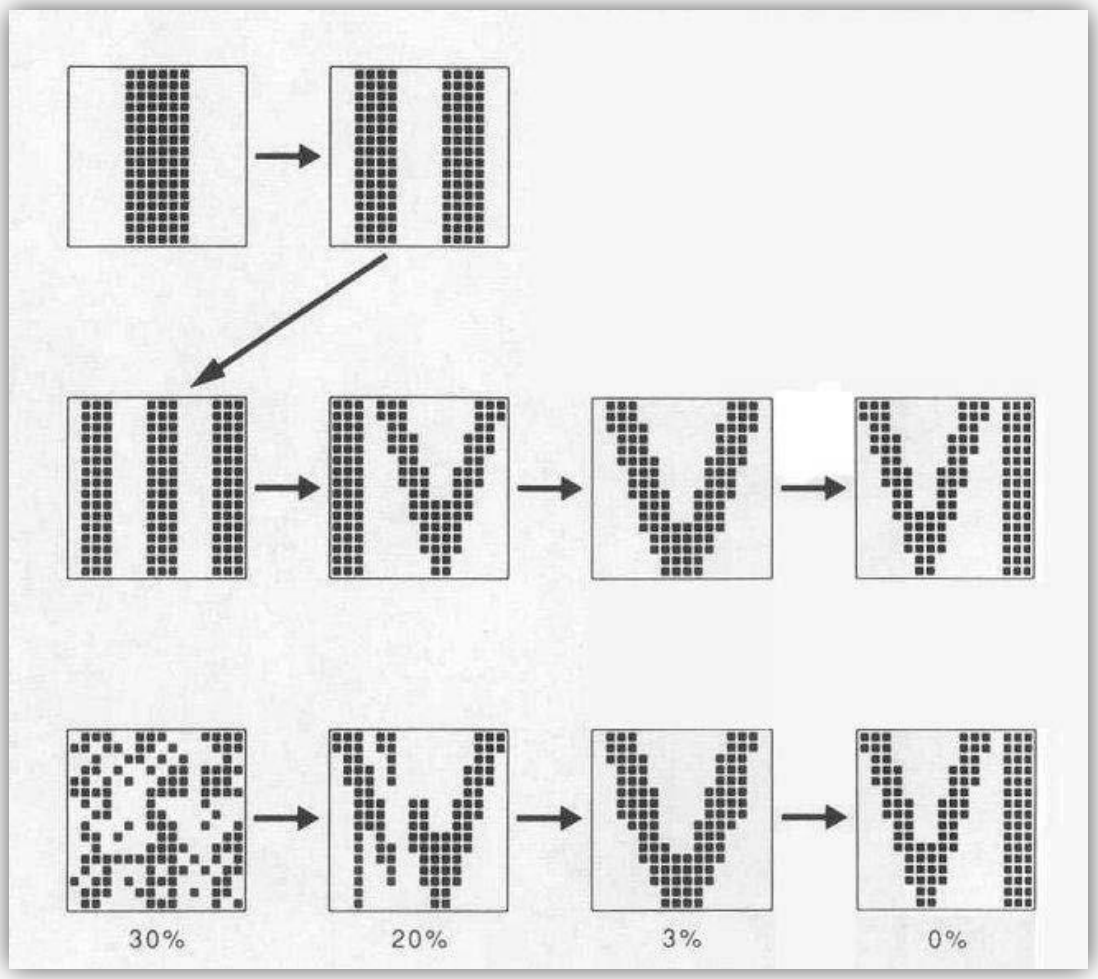


Fig. 7 Correlation of patterns in a Sparse Distributed memory, taken from "Kanerva's Sparse Distributed Memory, 1988".

Regarding the capacity of the SDM model, James Keeler has shown that the binary Hopfield Network and the sparse distributed memory models have the same capacity per storage element (0.15n) (30).

2.4 BAM - Bidirectional Associative Memory

The Bidirectional Associative Memory was developed by B. Kosko as an extension on the previously described Hopfield auto-associative memory (31). The major difference between both models resides in the fact that the Hopfield model contrary to Kosko's, is not bidirectional.

Kosko major objective was to find the minimal two layer nonlinear feedback network. As he states in his paper (31), information passes forward from one neuron field to the other by passing through the connection matrix *M*. In the same way, it passes backward through the

transpose of matrix *M*. In a BAM system, a set o pattern-pairs (A, B) are stored. To recall B, A is supplied as input to the system. For recalling A, the system takes B as input.

Let $\{(A_1, B_1), (A_2,B_2), ..., (A_p,B_p)\}$ be *p* training pairs. In order to encode these training pairs a correlation matrix is constructed as (31):

$$M = \sum_{i=1}^{p} A_i^T B_i$$

Where A^T is the transpose of A.

This correlation matrix defines the weights of a BAM. In such a structure if X_i^0 is applied as input, the bidirectional process is as follows (32):

$$\begin{aligned} F(X_i^0 M) &= Y_i^0 \rightarrow F(Y_i^0 M^T) = X_i^1 \\ F(X_i^1 M) &= Y_i^1 \rightarrow F(Y_i^1 M^T) = X_i^2 \\ &\vdots \\ F(X_i^j M) &= Y_i^j \rightarrow F(Y_i^j M^T) = X_i^{j+1} \end{aligned}$$

This represents the core behavior of the bidirectional associative memory.

Figure 8 presents an example of a BAM retrieval process[2]. The first *patterns* (upper left) were stored and associated in the memory. The corrupted patterns (bottom left) were then presented to the memory. The numbers near the arrows represent the number of iterations.

As can be seen BAM is tolerable to data corruption.

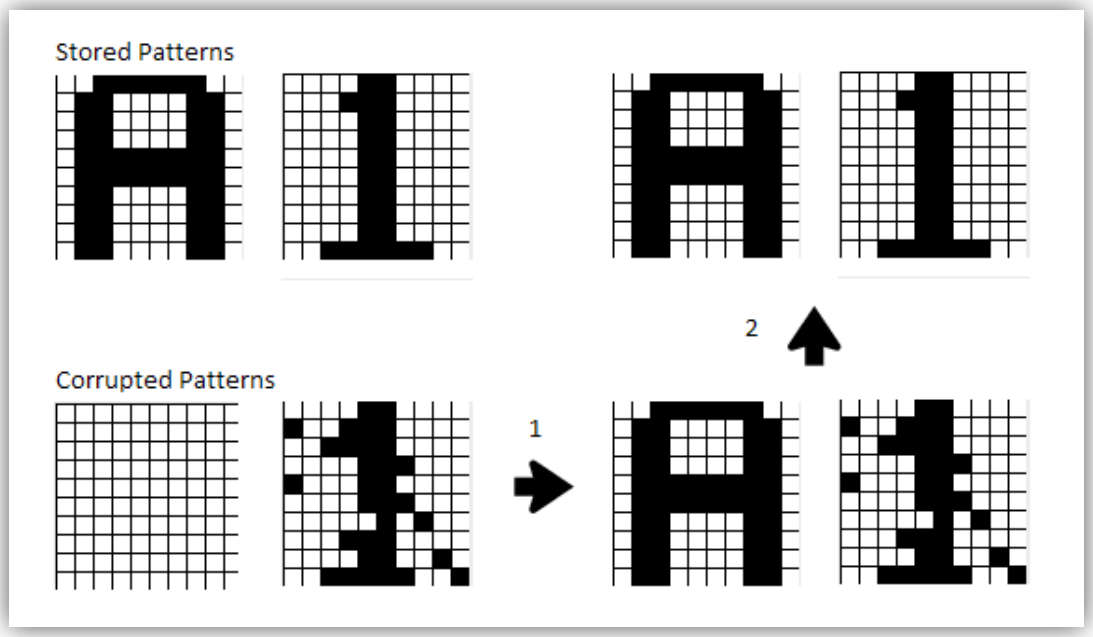


Fig. 8 Iteration process of a Bidirectional Associative Memory on corrupted patterns.

[2] **Artificial Neural Networks Research Lab.** *BAM Applet*

Referring to BAM's capacity, considering that n is the number of units of the binary vectors; it has been calculated to be 0.199n (33) (34).

2.5 Applications

The previous chapter described, theoretically, some of the most important associative memory models. In the following paragraphs, some of the current applications of those concepts will be presented.

2.5.1 Associative Memory for text documents

Introduction

Probably one of the first applications of associative memories was in the categorization of text documents. The objective of this solution is to find the most similar documents (35). An associative memory will assist to the classification of this similarity, and inputted documents can be stored in the memory according to their categorization.

As usual, the memory will be defined as a matrix, and logic operations will be executed in its columns. The logic operations that permit the classification will be implemented in hardware allowing for a faster execution.

Detailed Description

The implemented system consists of a "query by example" protocol; given an inputted document, that includes the type of content sought, the system will find the most similar documents. In order to do this some similarity rules will first have to be defined.

If the objective is to recover some information based on its content instead of a query key or memory address, then we will have to find a way to represent the data of each document (36). Before applying the pattern recognition methods, the text will have to be translated into *features.* In this case, there are two classes of features, one that expresses the spelling, and the second featuring the meaning and interpretation of each word. The semantics similarity is made through a class containing a table of relations, representing each word in a group of concept classes. This is possible through the coding of the text into certain trigrams. Assuming 32 possible characters, it can be calculated 32^3 possible trigrams; which demands a bit string of the same size. Each trigram will have a number assigned, and if the document contains a certain trigram, then its correspondent bit will be set to one in the feature vector. The result will be a binary vector with a size 32^3 containing ever trigram present in the document (35).

Another possibility is to combine different representations of the same text. In this case trigrams and digraphs would be used. For instance, consider the word "extension". It contains the following monograms:

"e" + "x" + "t" + "e" + "n" + "s" + "i" + "o" + "n" ;

And the following digraphs:

"ex" + "xt" + "te" + "en" + "ns" + "si" + "io" + "on";

Etc.

It is important to note that after the coding of the text, some information is lost on how many times a letter or digraph occurs, and in which order they were in the respective word. However, if the features are distinct in two different feature vectors, it can be concluded that the matching words will also be different; but if they are similar this does not imply that the text is identical.

There is also another class of features that can be used to classify text. Semantics, the meaning of the words, can also be used for their classification. A way to implement this type of categorization is to make a compilation of lists of words that are similar or identical in meaning, and then conclude that they belong to the same semantics *class* (36).

As an example consider the following words:

- Chair;
- Seat;
- Sofa;

All of the above words can be grouped in the same semantics *class* in the way that they all represent objects where you can sit.

Table 1 presents a group of five different semantic classes. These classes will be used to describe the text below.

Objects where you can rest (Class 1)	Objects that facilitate eating (Class 2)	Countries (Class 3)	Vegetables (Class 4)	Planets (Class 5)
Chair	Spoon	Canada	Lettuce	Earth
Seat	Fork	United Kingdom	Carrots	Mars
Sofa	Knife	France	Spinach	Venus

Table 1 Classes representing different semantic groups.

"I am sitting in my chair, eating a carrot with a fork".

If we were to calculate the feature vector for this text we would achieve the following result (consider that class 1 corresponds to the first position of the vector):

1 1 0 1 0

This is a rather simple example of the classification mechanism based on semantics. The classes and their own organizational scheme will depend on the settings that were chosen. There may be classes formed through the compilation of synonyms and others that contain words that are related on an abstract level; for example, in the same general area as economy, sports, science, etc. It is important to comprehend that the different codification methods described above can and should be combined, forming high dimensional feature vectors.

The recall process on the associative memory is defined by three different steps. The first step consists of discarding any non-relevant documents. This operation is similar to the one used in the *Lernmatrix* which was described in the beginning of the chapter. The application of the

inputted feature vector in the matrix will result in a group of candidates and correspondent measure of similarity (35). Based on this measure, it is possible to obtain a sub-group of candidates with the greatest similarity. This is the second phase.

The third and final phase consists on the textual analysis of both the inputted vector and the candidates of the sub group with greater similarity. This is an optional, verification phase.

As was explained before, the documents are stored through their feature vectors. Large documents should be divided into pieces of defined size, which will also be represented by feature arrays. A document can be seen as a hierarchical structure of smaller objects. Sections may be divided into paragraphs and paragraphs into sentences. It would be interesting to find such a small sub-division that the content of each part would be more or less unique, with only a single idea or topic. A page, by itself, is a cluster of fragments, whose length is a parameter selected by the user.

This is one of the most important factors to be defined, the size of the section to be represented in the associative memory. The search accuracy will be higher, the smaller the pieces are stored in the memory.

2.5.2 Associative Memory in face recognition

Introduction

The following application is a new technique on the classification of facial recognition (37). Face recognition has been one of the most active research topics these past decades. Given a set of subject face images as training samples, a face recognition system should identify a specific subject by an unknown face image. A subject to be identified from a facial image is previously classified and represents a class. The face recognition engine must determine whether or not an instance face image belongs to each class. The solution that will be described was achieved through the usage of genetic algorithms. Briefly, these algorithms have two purposes:

- To perform the fusion and selection of features for facial recognition;
- To locate the face areas which have the greater meaning in their classification;

Subsequently the features are organized in a vector and compared with the data stored in the associative memory.

Detailed Description

The common description of a facial recognition system describes it as a computer application for automatically identifying or verifying a person from a digital image. The most usual way to achieve this is by comparing selected facial features between the image and a facial database. The proposed solution has a similar behavior to this description (37). An associative memory is used to store and classify the similarity between images. The process itself is similar to Steinbuch's associative memory (12). In the learning phase each feature vector is stored in the memory. In the retrieval phase a feature array is presented to the memory and the output will be the most similar vector. The following paragraphs will, briefly, describe how these features are obtained. Figure 9 presents a basic diagram of the retrieval phase.

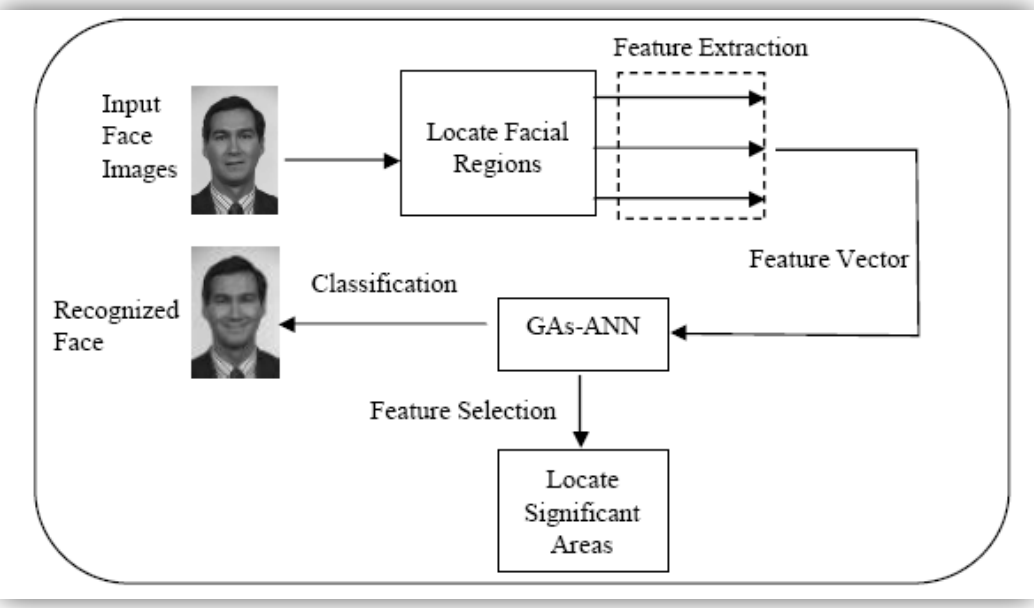


Fig. 9 Block Diagram of the methodology in a facial recognition engine.

The first step in the process is to locate the facial regions that are significant for classification; the left and right eye, the nose, mouth and chin (37). The description of the process in which each region is located is beyond the scope of this review.

After the location of the facial regions, each one is sub-divided into rectangles. These allow for the extraction of data related to the chromosomes, which represent the facial indicators of the person. Each sub-area has a size of 6x4 (37). After the sub-division, it is extracted and associated the average gray value of each area. In the end of the process the output will be a vector with 90 different features that describe the face of the person. This is the pattern that will be presented to the associative memory in the retrieval phase.

2.5.3 Associative Memory in gesture recognition and training

Introduction

The following solution is a computational method for classification of gestures (38). In general, a gesture is considered a movement initiated by the hand or body to manipulate a cursor in order to create a pattern in a certain direction, during a certain period of time. This application allows a computer system to accept input data, generated by the user, and interpret the gesture made by the cursor.

In this solution the pointing mechanism is a computer mouse, but any kind of device with a cursor, including pen, trackball could be used with the same degree of success. The catalyst for the initiation of the process is the pressing of a key in the keyboard. After the process starts it is recorded the position of the cursor throughout the time. Afterwards it is determined a feature vector based on the captured information. The feature array is then presented to the associative memory that classifies it and returns the most similar gesture previously stored. The returned gesture and every operation associated with it are then executed.

Detailed Description

The implemented system has the objective to capture and classify gestures produced with a cursor. In order to achieve this objective it is used an associative memory, which is responsible for the storage and classification of the gestures. The process initiates with the learning phase. Every gesture is associated and stored in the memory. A gesture is characterized by multiple strokes. A stroke is represented by various segments. These segments are represented by its size, direction and time (counted from the initial starting point until the point where the segment ends) (38). This information is then organized into feature vectors that are then presented to the memory.

Figure 10 presents a retrieval example on the associative memory for gesture recognition. As can be seen the property that allows the correction of faults if information is missing or invalid in the input pattern is present in this application.

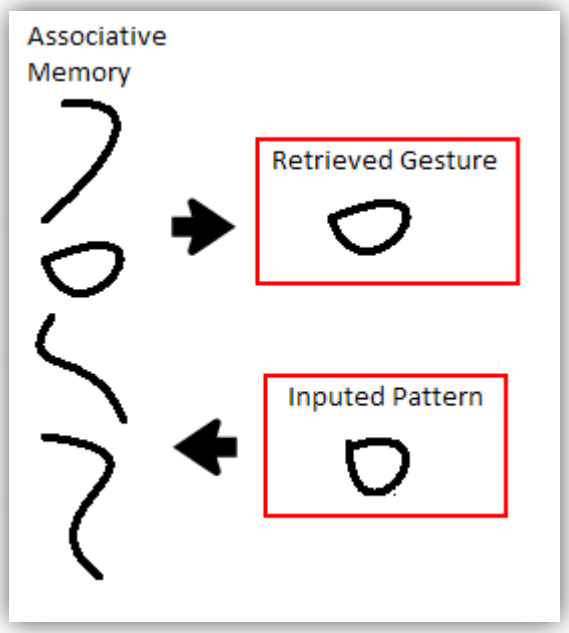


Fig. 10 Retrieval phase example in a gesture recognition system.

2.5.4 Associative Memory in voice recognition and training

Introduction

Another topic which has been very active lately is voice recognition. Speech is a natural mode of communication for the human being. It comes so naturally that we don't realize and understand how complex it is. The human vocal tract is a biological organ with non linear properties, which is affected by very different factors like gender and emotional state (39). All of these variations make speech recognition a complex problem. Furthermore it is very difficult to accurately evaluate a voice recognition system. Below are some of the different characteristics of these systems that should be evaluated in order to calculate their efficiency (40) (41):

- **Vocabulary size**: The size of the vocabulary that we are trying to recognize has great impact on its efficiency. For example, vocabulary sizes of 100, 1000 or 10000 words

may have error rates of 1%, 8% and 56%. On the other hand, even small vocabulary can be hard to recognize if it contains words that are similar.

- **Speaker dependence**: A system can be intended for use by a single speaker or for use with any speaker. Speaker independent systems are hard to develop because a system's parameters become tuned to the speaker it was trained on.
- **Isolated, discontinuous or continuous speech**: Recognition efficiency depends largely on the kind of data that is inputted into the system. Isolated speech, a single word, is usually easy to process. Discontinuous speech (a full sentence) can also be easily recognized by a previously trained system. A continuous stream of speech is difficult to recognize and its degrees of success, variable.
- **Language constraints**: Different languages have different constraints on the word sequences that are allowed during recognition. Constraints are often represented by a grammar which will filter incorrect sentences so that the recognizer engine can evaluate the remaining data correctly. Usually, it is easier to measure the difficulty of a recognition task by the grammar perplexity (number of words that can follow any given word) rather that by its vocabulary size.
- **Read and spontaneous speech**: Systems that are previously trained on a prepared script achieve better results than when they are presented to spontaneous speech. Spontaneous speech presents some characteristics that make recognition vastly more difficult, like: incomplete sentences, coughing or stuttering.
- **Adverse conditions**: Common knowledge dictates that different microphones, acoustical distortions or environmental noise affect the systems efficiency in a negative way.

The following chapter will present a solution for a speech recognition system using an associative memory for classification (42). The purpose of this system is to allow experimentation of an associative memory driven voice recognition system in a real case scenario.

Detailed Description

A standard voice recognition system, both in training and recovery phases, contains most of the computational complexity concentrated in the pattern recognition sub-system. Speech recognition is a multileveled pattern recognition task in which acoustical signals are structured into words, phrases and sentences. Additionally most of these interpretations have temporal constraints. The structure of a standard speech recognition system is presented in Figure 11.

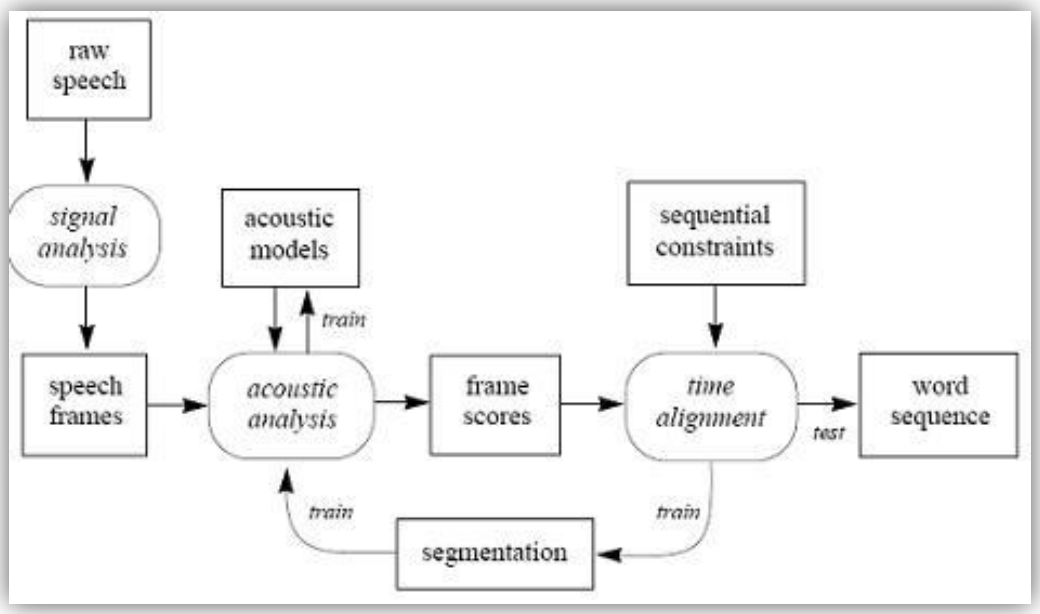


Fig. 11 Architecture of a standard speech recognition system.

Raw speech is the voice that is sampled at a high frequency over a microphone. Signal analysis consists on the transformation of the initial speech, in order to simplify its processing. There are many different techniques that permit this compression but its description is out of the scope of this thesis. The result of the signal analysis process is a sequence of speech frames. Figure 12 presents an example of this transformation.

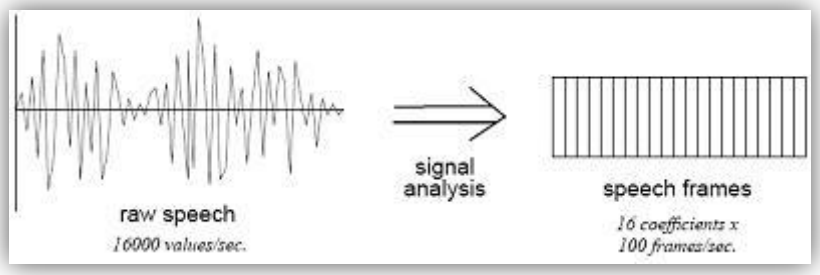


Fig. 12 Diagram illustrating the transformation of raw speech into frames.

In a wireless communication system, for instance a wireless telephone, the user's voice signal is received through the microphone of a mobile device. The analog signal is then digitized in order to produce a stream, for example 8-bit samples per second. Trying to send these samples through a wireless channel is very inefficient; hence the information is usually compressed before transmission. Through a technique called *vocoding*, the voice stream is compressed in a range of packages. These smaller packages are sent through the wireless channel instead of the voice samples they represent. Subsequently, the base station receives and unpacks them in order to produce the respective voice samples.

Vocoder's main purpose is maximum compression maintaining however the ability to understand speech (43). These algorithms are *lossy* in a way that, on the decoding side, the information is not exactly the same as conveyed by the speaker. Furthermore, these

algorithms are optimized to maintain the understanding of speech, even if some packets are lost during transmission. This optimization also leads to a greater mismatch between the information sent and received, although the degree of degradation varies widely between algorithms.

In this solution, the mobile device performs the extraction of acoustic features and transmits the feature vector instead of *vocoder* packages. Because these patterns occupy less bandwidth than vocoder packages, they can be transmitted through the same channel with additional protection in order to minimize channel errors. Each feature vector is extracted through a series of voice samples, collected over a fixed interval. Alternatively these ranges may overlap. For example, the features may be obtained at 20 millisecond intervals, starting every 10 milliseconds, so that two consecutive intervals share the same 10 millisecond segment (42).

On the receiving side, the vectors are processed by an adaptation engine that performs the selection of a function corresponding to the user. This function is applied to the receiving data, and the result is a pattern that is adapted to each user that previously trained the system. In this regard, the system is speaker independent as long as each user spends some time training the engine.

Another interesting property of the system is that it performs a dynamic classification. In static classification the memory sees all of the input speech at once, and makes a single decision. By contrast, in dynamic classification the network sees a small window of speech. Static classification works well for phoneme recognition. Dynamic classification scales better for spontaneous speech.

2.5.5 Associative Memory in music classification

Introduction

Associative memories and neural networks have achieved much success in the recognition of patterns. Through the classification of inputs into groups, the network can be trained to discern the criteria for classification, and generally allow the classification of inputs not used during training. With the *explosion* of digital music in recent years due to *Napster* and the internet, the application of technology for recognition of standards for digital music has become increasingly interesting. The following solution presents a way to, given an artist or musical genre, identify similar music previously stored in the memory (44).

Detailed Description

From a user point of view, it is overwhelming to imagine how many people have downloaded music files (mp3, wav) that are at the moment stored in their hard drives. It is then easy to understand the usefulness of a program that allows to rate and classify new downloaded music according to the user criteria. Another interesting feature would be a program that searches for a group of files and extract only those features that were defined previously. For instance, consider a user that wants to search for classical music in a computer in Austria, but due to language differences finds it difficult to execute a query by name. A program that could carry out this classification and search through the contents of the music would be much more appropriate and useful (45).

With Napster and the music industry, the classification based on musical content is necessary to ensure that there are no copyrighted infringements. Filters based on file names were considered inefficient because more experienced users change the names of files to bypass these filters. The purpose of this project is to study the reliability of using a classification system based on content that relies on a neural network for classification. Figure 13 represents a block diagram of the classification system.

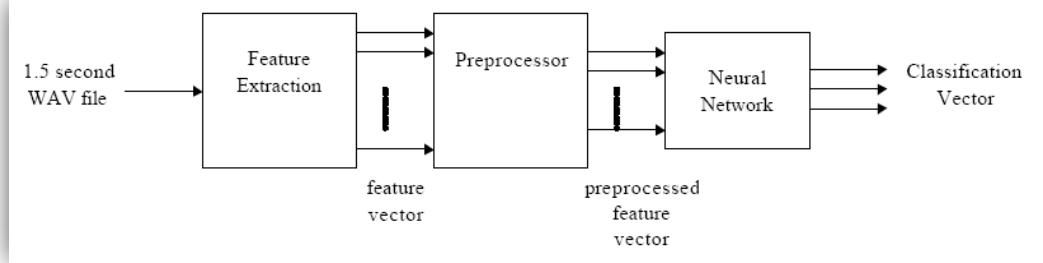


Fig. 13 Classification system on an associative memory for music classification.

A file of 1.5 seconds in wav format is passed to a function that extracts features based on its content. This engine calculates 124 numerical features that characterize the sample. During the training phase this process is carried out with different wav files in order to populate the associative memory.

Ideally, the samples in the wav file would be passed to the neural network and then be determined how best to process the data and classify the file. However, with a sampling rate of 44.1 kHz, even a second of audio would result in a prohibitive amount of information. It is then necessary to have a feature extraction function that reduces the amount of information presented to the network. The extraction of the best features of a song is a very popular subject in research departments. These features were narrowed from the potential thousands to 124 (see Figure 14).

Feature Numbers	1-32	33-64	65-96	97-108	109-123	124
Feature Type	LPC Taps	DFT Amplitude Value	Log of DFT Amplitude Values	IDFT of Log of DFT Amplitude Values	Mel-Frequency Cepstral Coefficients	Volume

Fig. 14 Description of the 124 features used in music classification.

The feature patterns returned by the extraction feature process were first pre-processed before being delivered to the neural network. Two types of pre-processing were used; one to scale down the data to comply with a range between -1 and 1, and another to reduce the size of the input vector. The data was divided into three sets, one for training, one for validation and one for testing. The pre-processing parameters were determined using a matrix that contains all feature patterns used for training and validation.

It was used two classification systems. The first concerns classification by genre (rock, pop etc.) (41). The second classification system revolves around the artist that composes the music. The results show that the classification by artist is a task far more complicated than classification by genre. The reason for this is that artists sometimes change the characteristics that allow the classification; musical instruments, rhythm and volume. In some cases it is assumed that a larger feature set would vastly improve the classification efficiency.

Chapter III

Associative Memory - Capacity

This chapter will address the reason why this project focuses on Steinbuch's associative memory instead of other, often more recent, models. As was said before chapter I, the storage potential of the *Lernmatrix* is greater than other models if the optimal sparseness value is used. The following paragraphs will demonstrate the capacity value of Steinbuch's associative memory and compare it with the associative memories described in chapter II. The values illustrated below will be considered in this proof:

- Bidirectional Associative Memory model capacity is 0.199n, where *n* is the binary vector's size (33) (34).
- Hopfield's model capacity is 0.15n (11).

Consider the following formula that represents the number of different binary vectors of dimension *n* with *M* ones.

$$(1)\ C(n,M) = \frac{n!}{(n-M)!\,M!}$$

As an example consider the following set of binary vectors where the dimension is 3.

001
010
100

Introducing the *n* and *M* values in the formula will result in:

$$C(3,1) = \frac{3!}{(3-1)!\,1!} = \frac{3\times 2\times 1}{(2\times 1)\times 1} = 3$$

This is true considering the set of vectors presented above.

It is also possible to determine the probability of presence of a randomly selected binary vector. Consider the following:

$$(2)\ P_{C(n,M)} = \frac{1}{C(n,M)}$$

In order to calculate the capacity of the *Lernmatrix* it is necessary to introduce the concept of entropy. Entropy is a concept introduced by Claude Shannon in 1948 that measures the uncertainty associated with a random variable (46). Shannon's entropy is defined in the following formula:

$$(3)\ I = H(x) = \sum_{i=1}^{n} p(x_i)I(x_i) = -\sum_{i=1}^{n} p(x_i)\log_b p(x_i)$$

H is considered the entropy and *I* is the information content of *x*. Consider that *L* is the number of vectors to which the entropy will be calculated.

$$(4)\ H(x) = -L \times p(x_i) \log_b p(x_i)$$

Replacing the probability factor for the probability of a random vector will achieve the following:

$$(5)\ H(x) = -L\ P_{C(n,M)}\ \log_2 P_{C(n.M)}$$

Combining (2) with (5) and applying the correspondent logarithmic properties it is possible to obtain the formula:

$$(6)\ H(x) = L\ \frac{1}{C(n,M)}\ \log_2 C(n,M)$$

Maximizing the information in relation to the size of the associative memory and replacing (1) in (6):

$$(7)\ I = L\ \log_2 \frac{n!}{(n-M)!\,M!}$$

The capacity will depend on the size of the binary vector, *n*. As the objective is to find the optimal capacity, it is necessary to find a way to calculate the optimal values for *M* and *L*. Consider that after storing some binary vectors in the memory, *p* is the probability that a weight at a certain position is one, and 1-*p* the probability that the weight is zero. The probability, for one pair, that a weight is zero corresponds to the following:

$$(8)\ \frac{(n-M)}{n} \times \frac{(n-M)}{n}$$

Considering *L* pairs, and the probability property that indicates that the probability of an independent sequence of events is the product of each event individual probability, it can be concluded:

$$(9)\ (1-p) = \left[\frac{n^2-M^2}{n^2}\right]^L \leftrightarrow p = 1 - \left[1 - \frac{M^2}{n^2}\right]^L$$

As was determined before, a vector *x* with *M* ones has a probability *p* of a weight being one. Consequently the probability of getting a wrong output is p^M. Consider that the number of wrong ones on each recalled vector is exactly 1. Combining all these factors with the number of zeros in a given vector will result in the following formula:

$$(10)\ (n-M) \times p^M = 1$$

This represents the number of zeros in the vector and the probability of each zero being wrongly set to one. Replacing (10) in (9) will present the optimal value for *L*.

$$(n-M)\left[1 - \left[1 - \frac{M^2}{n^2}\right]^L\right]^M = 1 \leftrightarrow 1 - (n-M)^{\frac{1}{M}} = \left[1 - \frac{M^2}{n^2}\right]^L$$

$$(11)\ \log(1-(n-M)^{\frac{1}{M}}) = L \times \log\left[1-\frac{M^2}{n^2}\right] \leftrightarrow L = \frac{\log(1-(n-M)^{\frac{1}{M}})}{\log\left(1-\frac{M^2}{n^2}\right)}$$

Combining (7), (9) and (10) will result in:

$$I = \frac{1}{\log(2)}\left(\frac{\log(1-[n-M]^{-\frac{1}{M}}}{\log\left(1-\frac{M^2}{n^2}\right)}\right)\left[\left(n+\frac{1}{2}\right)[\log n - \log(n-M)] + M\log(n-M) - \left(M+\frac{1}{2}\right)\log M - 0.92\right]$$

In order to find out the optimal value for *M* a computer will be used to maximize I/n^2 for each value of *n*. This computation can be seen in Table 2.

n	*M*	I/n^2
10^2	5	0.54
10^3	8	0.55
10^4	11	0.57
10^5	14	0.58
10^6	18	0.59
10^7	21	0.60
10^8	24	0.61
10^9	28	0.62
10^{20}	64	0.63
10^{80}	263	0.67
10^{100}	330	0.68

Table 2 Computational result of the *Lernmatrix* storage capacity after maximizing *I*.

From the information above and the experiments made by Robert Hecht-Nielsen in *Neurocomputing* (11) it can be concluded that the optimal value for *M* is:

$$M = \log_2\left(\frac{n}{4}\right)$$

By substituting the value for *M* in the equation (10) it is possible to calculate the capacity of the *Lernmatrix*, which is:

$$L = \ln(2)\left(\frac{n^2}{M^2}\right)$$

If the optimal *M* value is used, the number of vector pairs that can be stored in the associative memory will be much greater than the size *n*. Comparing the *Lernmatrix* storage capacity with the other models will show its merit. Consider a unit value of 8(*n*). In order to calculate the optimal number of ones presented in the vector we will do the following:

$$M = \log_2(2) \leftrightarrow M = 1$$

The storage capacity of Steinbuch's associative memory for a size of 8 is:

$$L = \ln(2)\frac{64}{1} \leftrightarrow L = 44.4$$

The capacities of the other models are as follows:

- Bidirectional Associative Memory model: $0.199 \times 8 = 1.592$.
- Hopfield's model: $0.15 \times 8 = 1.2$.

As can be seen, the asymptotic storage capacity of the *Lernmatrix* is far better than those of BAM and Hopfield models.

A better test case would be to consider a larger value for *n*. Consider a vector with a size of 10000 units. The optimal value for *M* is:

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

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:

- Bidirectional Associative Memory model: $0.199 \times 10000 = 1990$.
- Hopfield's model: $0.15 \times 10000 = 1500$

The results indicate that in a real case scenario with optimal values, Steinbuch's model presents a much bigger capacity than the other models. It is due to this storage potential that the *Lernmatrix* was the chosen research framework.

Chapter IV

Methods

This chapter will explain the mathematics that is behind the solutions to both problems presented in the objectives chapter. 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. These formulations will be used and tested in Chapter V.
- **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.
- **Presentation methods:** the results chapters will present some information regarding the distribution of ones in the memory. The way to achieve this distribution will be shown in this section.
- **Pointer Representation:** the data regarding the associative memory, its inputs and outputs will be presented in pointer representation. This section will explain this structure.

4.1 Hierarchical associative memory methods

The following information is about the aggregation and hierarchy of an associative memory.

Consider a square associative memory with dimension *n* and *M* number of ones. Assume also that the memory will be implemented in pointer representation (section 4.4). Zeros are not presented in pointer representation and 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 values for *a* and *b* are:

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

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 following chapter.

4.2 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 high dimensional space. 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: 01010110 11101101.

For a window of size eight, the vector is divided in the following 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 is:

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

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 other 254 positions 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 problem of presenting non sparse data to the *Lernmatrix*.

4.3 Presentation methods

Most of the information in the results chapters will be presented in the form of graphics describing execution times, number of operations and errors.

Another kind of data that will be presented consists on the weight matrix diagram and structure of the weight matrix. These are important as they present a clear view on the state of the associative memory before each experiment.

Weight matrix diagram

This diagram illustrates the weight distribution that results after the storage of each pattern in the memory. This snapshot permits to verify if the weights are evenly distributed over the whole matrix, as this is the property that most favors its efficiency. The presence of clusters in the diagram suggests a high correlation between the stored patterns. Another important factor that can be observed is the load of the associative memory. The load is the percentage of weights that are set to one. A higher percentage is detrimental to its efficiency. It is to be noted that a one in the memory is represented by the white color and a zero by black. Figure 19 presents a matrix with evenly distributed weights.

Weight matrix distribution

The distribution of the weight matrix is another tool that permits to confirm the state of the associative memory. Briefly, it consists of the sum values of the weights of each row or column arranged in ascending order. The first step is the sum over each column, which is given by:

$$\mu_i = \sum_{j=1}^{n} \delta(w_{ij})$$

Where μ is the sum, i and j the columns and rows respectively.

$$\delta(x) = \begin{cases} 1 \; if \; x > 0 \\ 0 \; if \; x = 0 \end{cases}$$

The second step is to arrange each unit in ascending order:

$$\mu_1 \leq \mu_2 \leq \mu_3 \leq \cdots \leq \mu_m$$

The final step before plotting the values is to sum each similarμ:

$$\mu_1 = \mu_2 \; (2) < \mu_3 (1) < \mu_m$$

Annex D.7 represents the weight matrix distribution of the matrix depicted in Figure 19.

4.4 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 (47). In pointer presentation only the positions of the ones are represented. Consider the following binary vector: **00100010011**. In pointer format this vector is: **4: 2 6 9 10**. The first component consists of the number of ones in the binary vector and the following numbers the positions of each one. Figure 15 represents an input file containing 5 images of size 400.

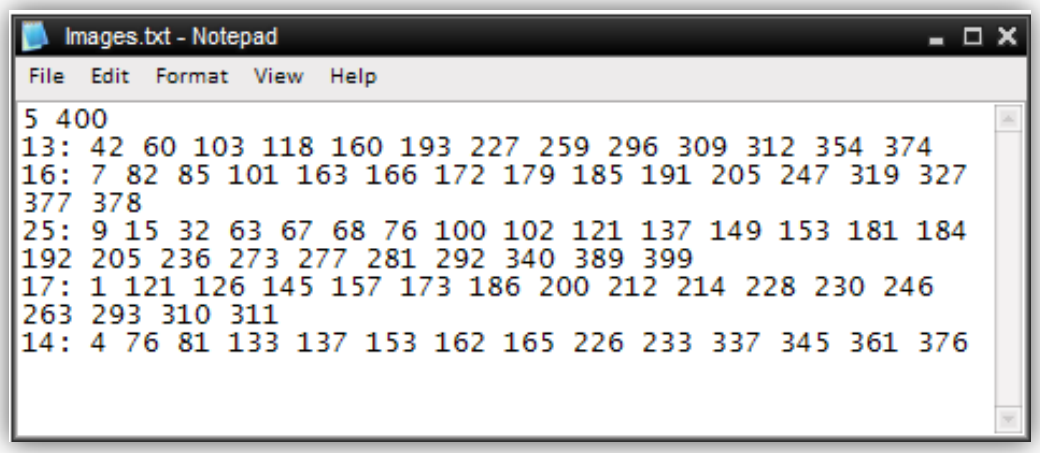


Fig. 15 Example of an input file in pointer representation.

Throughout the project, every input file will have this representation as it will save memory space and execution time. The disadvantage is that the vectors are harder to read and its utilization less intuitive.

Chapter V

Hierarchical associative memory results

In the previous chapter, it was demonstrated the mathematics behind the solutions to the problems of efficiency and presenting non sparse feature vectors to Steinbuch's *Lernmatrix*. This chapter presents the results regarding the experimentation of the method described in section 4.1. It relates to the inefficiency problem of the Lernmatrix when implemented over a serial computer.

The associative memory is a mechanism that has the objective to, given an input pattern, determine the most similar stored patterns. Besides this objective, there are several other advantages:

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

In this regard, this chapter presents an evaluation of the previous explained solution to decrease the execution time of the recall phase on an associative memory.

It is the purpose of the following paragraphs to show the results obtained after running an associative memory with the aggregation of its elements. This aggregation will permit to find out in a previous, less intensive step, if there is matching pattern. Figure 16 illustrates the aggregation method in both the input vector and associative memory. In this case, the aggregation factor is three.

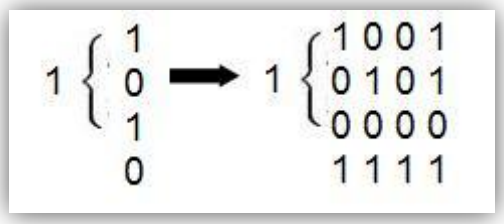


Fig. 16 OR Aggregation example in hierarchical associative memory.

The consequence of this aggregation is that as long as there is a one in the aggregation set, the outcome will be one. This will reduce the size of the associative memory and input pattern and consequently the time it takes for the associative memory to find if there is a matching pattern. Figure 17 presents an example of this aggregation.

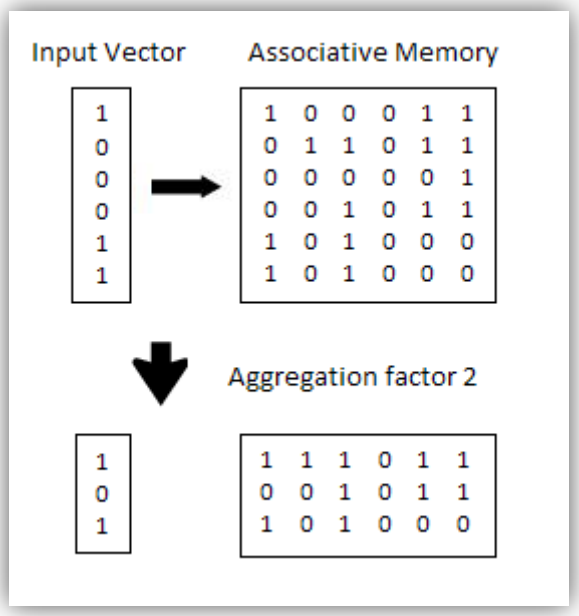


Fig. 17 Aggregation process on the recall phase of hierarchical associative memory.

This is a previous step, before the *normal* associative memory acts; before going through the larger associative memory, the aggregated one will indicate if a matching pattern can be found.

The next chapters will show the results obtained after some experiments with this method. 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 chapter III. 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.

5.1 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 18 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.

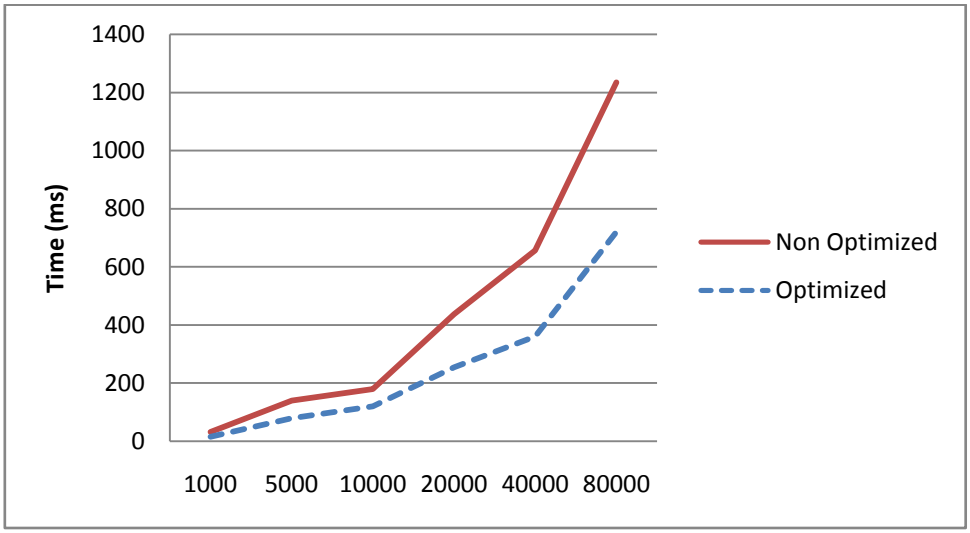


Fig. 18 Execution Time with Aggregation 2 (no correlation).

It is important to understand the state of the memory after each test. In this regard the following section will present the weight matrix diagram of the memory and its distribution according to the methods described in chapter IV. This information will allow to verify if the storage was executed successfully.

5.2 Associative memory diagrams (no correlation)

One thousand images

The figure in Annex D.1 represents the diagram of the associative memory after storing 1000 images.

For each position, the weight of a one is represented in a white color and a zero in black. As can be verified, the memory is not full. In fact it is almost empty. This can be explained with the lack of correlation in the storage phase.

It can be verified that the storage is correct and that the data is randomly generated. Figure 19 illustrates the distribution of ones in the memory and confirms the statement above.

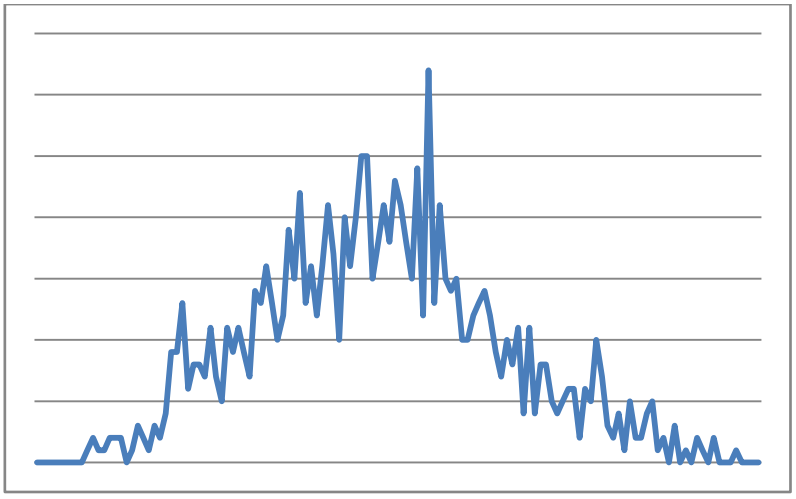


Fig. 19 Distribution of ones in the memory after the storage of 1000 images with no correlation.

Five thousand images

Annex D.7 illustrates a snapshot of the associative memory after the storage of 5000 images without correlation. Both Annex D.7 and Figure 20 show that the data is randomly generated and stored correctly. The image also indicates that the memory is getting fuller, as expected.

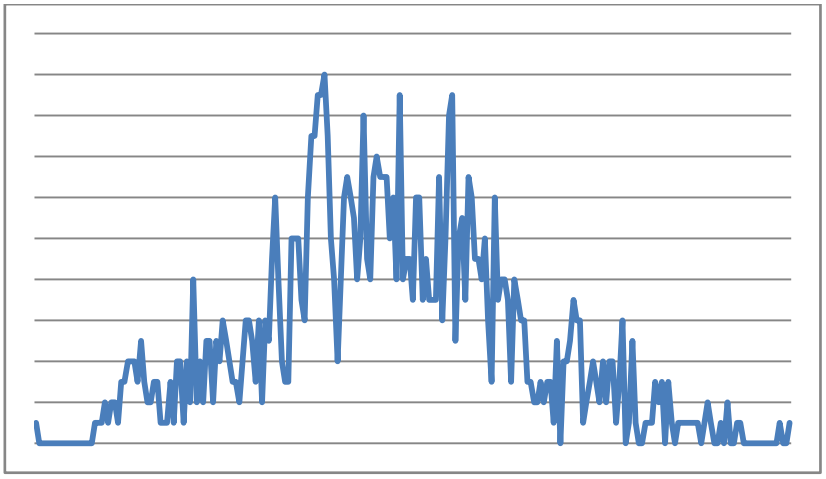


Fig. 20 of ones in the memory after the storage of 5000 images with no correlation.

Ten thousand images

In Figure 21 it can be clearly seen that the ones are evenly distributed across the memory. Once more this feature indicates that the data is randomly generated and correctly stored.

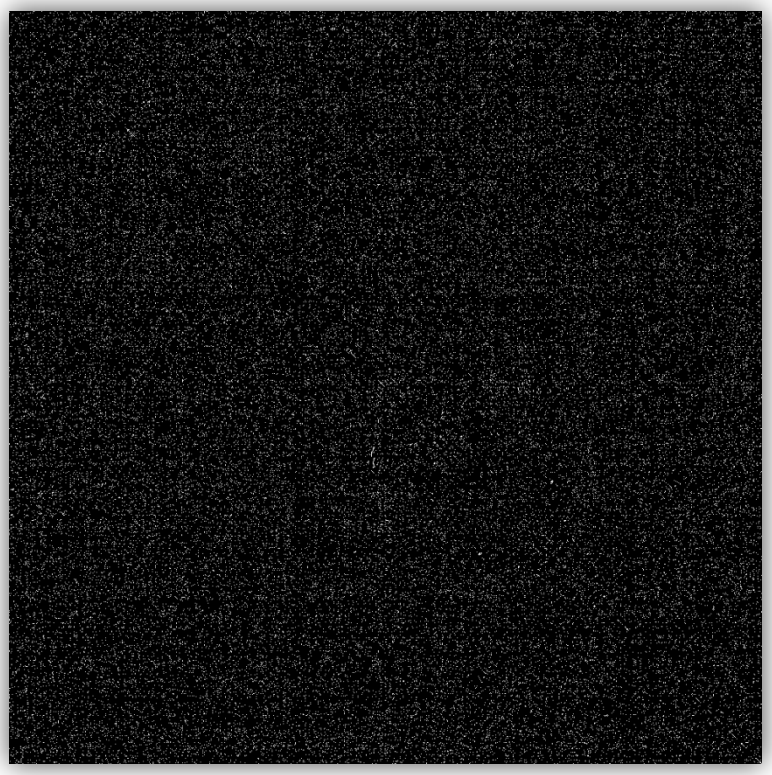


Fig. 21 Snapshot of the memory after the storage of 10000 images with no correlation.

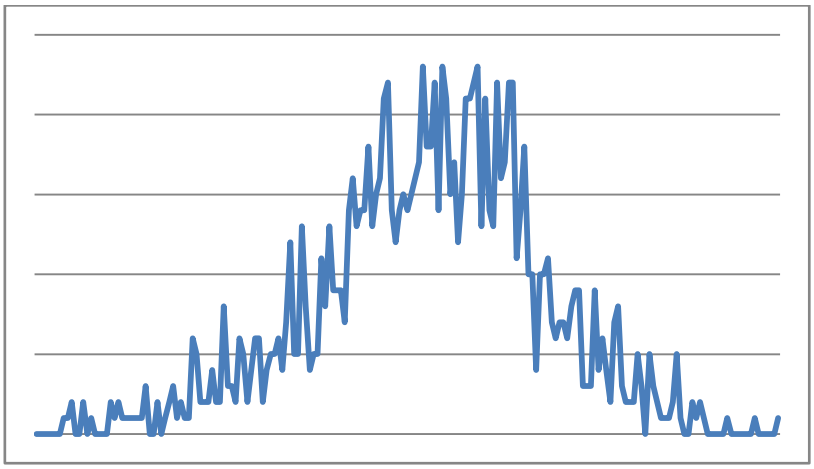


Fig. 22 Distribution of ones in the memory after the storage of 10000 images with no correlation.

Forty thousand images

The snapshot of the memory (see Figure 23) illustrates that it is almost full. This accounts for the higher execution time presented in Figure 18.

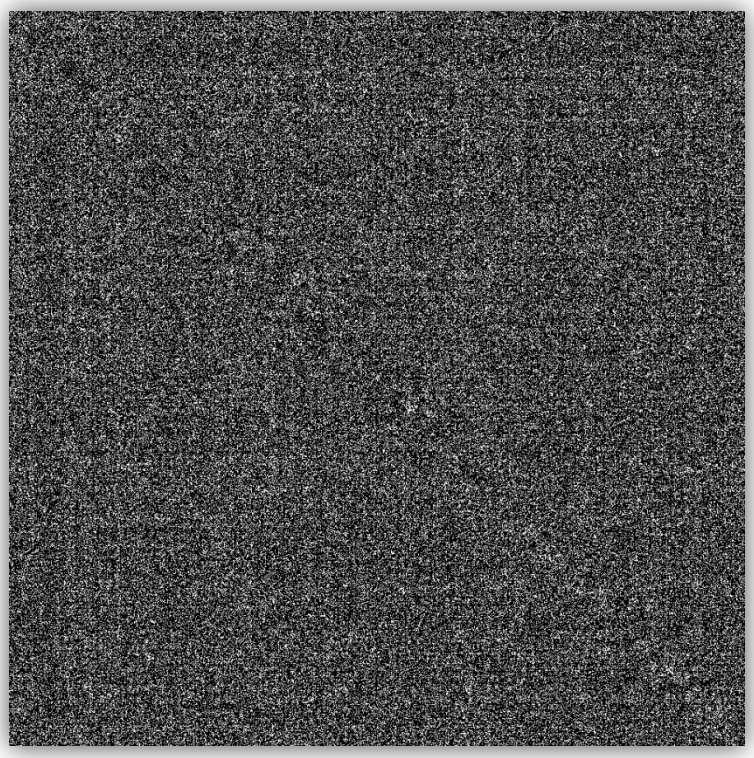


Fig. 23 Snapshot of the memory after the storage of 40000 images with no correlation.

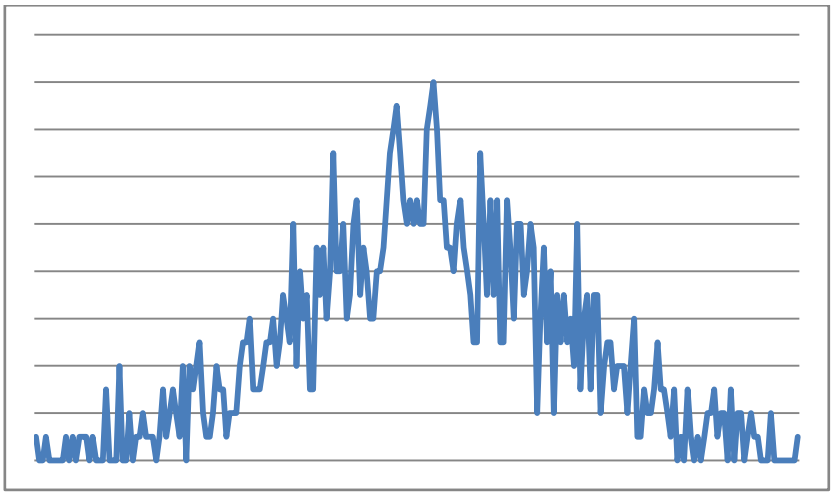


Fig. 24 Distribution of ones in the memory after the storage of 40000 images with no correlation.

The weight matrix diagrams corresponding to the experiments with 20000 and 80000 images will be skipped as it is redundant information.

These diagrams and distributions indicate that the datasets were correctly generated and stored; and that the associative memory is behaving as it should. They also confirm the veracity of the tests. The information depicted in this section is related to every experiment without correlation.

Aggregation coefficient 5

As the aggregation coefficient increases, we can only hope that the difference between the optimized execution time and the non optimized execution will grow larger. It is also important to understand that as the coefficient goes higher, so does the probability of finding a match in the associative memory. If a match can be found, then both the aggregated and non aggregated steps must be executed, in which case, the optimized memory execution will always take more time. The results (see Figure 25) indicate that there is only a small decrease in execution time between aggregation 2 and 5.

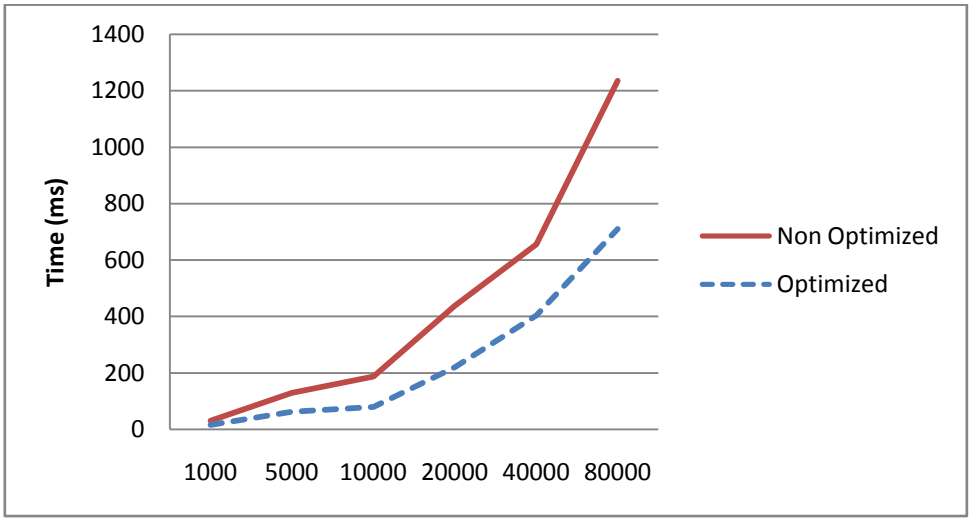


Fig. 25 Execution Time with Aggregation 5 (no correlation).

Aggregation coefficient 10

As Figure 26 illustrates, a higher coefficient does not imply a major decrease in the execution time.

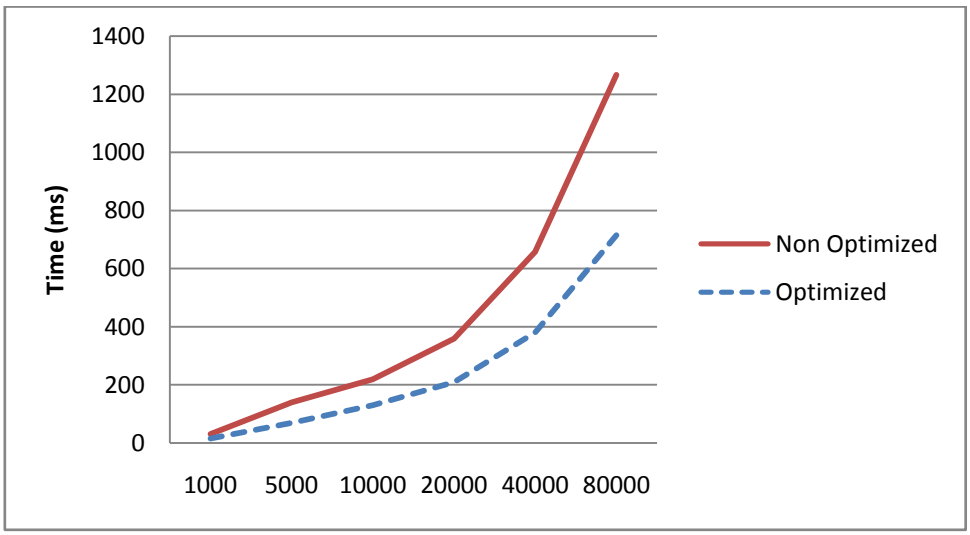


Fig. 26 Execution Time with Aggregation 10 (no correlation).

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 3 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.

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 3 Number of operations for each aggregation coefficient in hierarchical associative memory.

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 27 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.

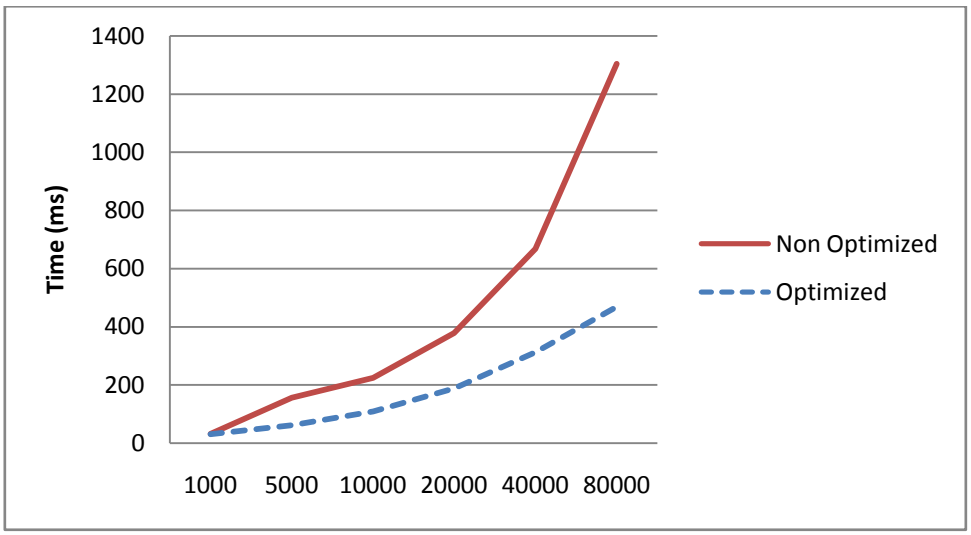


Fig. 27 Execution Time with a three step aggregation process (no correlation).

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.

5.3 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 28 are unexpected. As it seems, 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 aggregated and non aggregated steps will be performed.

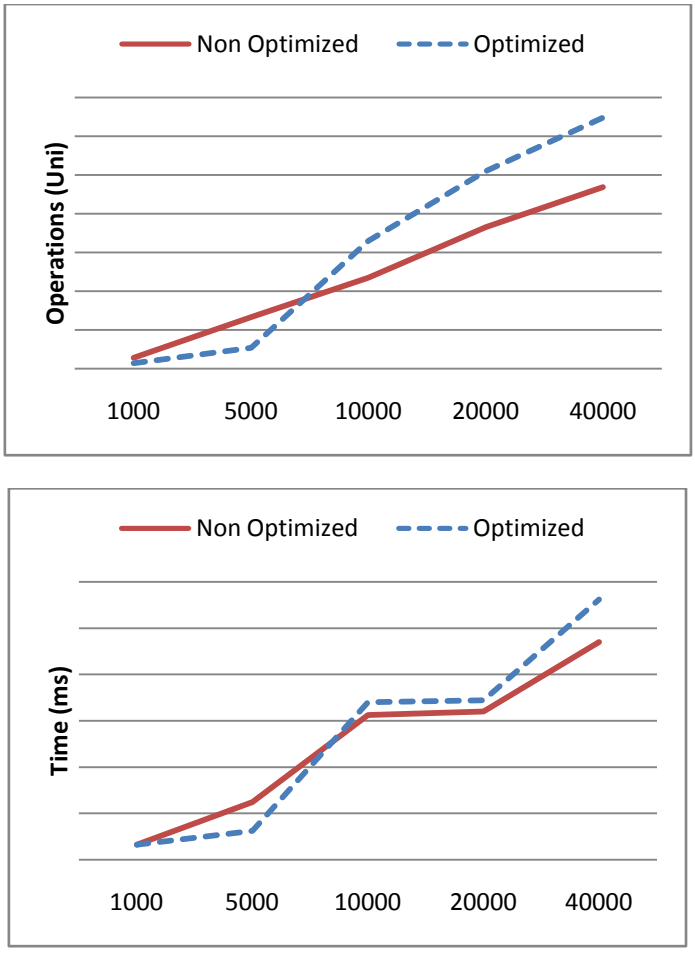


Fig. 28 Execution Time and number of operations for Aggregation 2 (with correlation).

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.

Similar to the non correlated experiments chapter, the following section will present the weight matrix diagram and distribution for each experiment.

5.4 Associative memory diagrams (correlation)

One thousand images

Comparing Annex D.8 with Annex D.1 (the corresponding experiment without correlation) it can be noticed the differences between both memories. First, with correlation, the memory is fuller. Second, the correlation can be easily verified by looking at the diagonal; which is filled with ones. Besides this, it can be verified that the patterns stored in the memory were randomly generated and that the storage was made correctly.

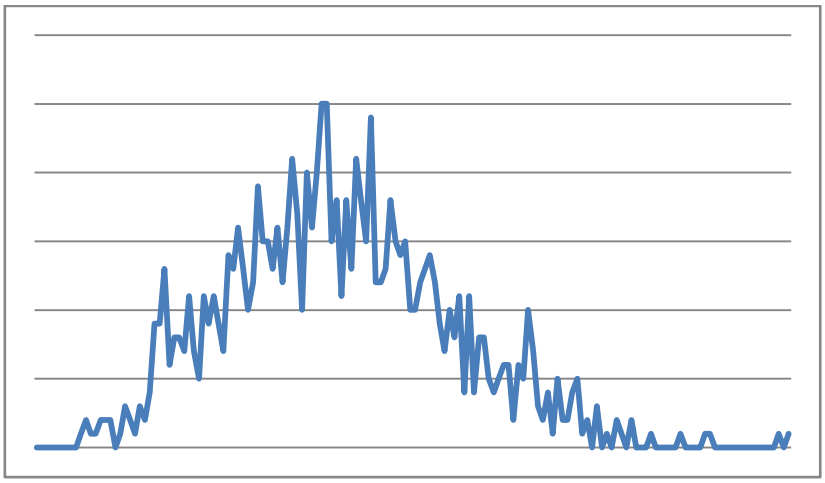


Fig. 29 Distribution of ones in the memory after the storage of 1000 images with correlation.

Five thousand images

Figure 30 indicates that the memory is getting fuller faster than without correlation. This may explain why after 5000 images the probability of finding a matching pattern is so high. As was explained before, if the probability of a *hit* is very high, the execution of the aggregation steps will slow down the overall average query time.

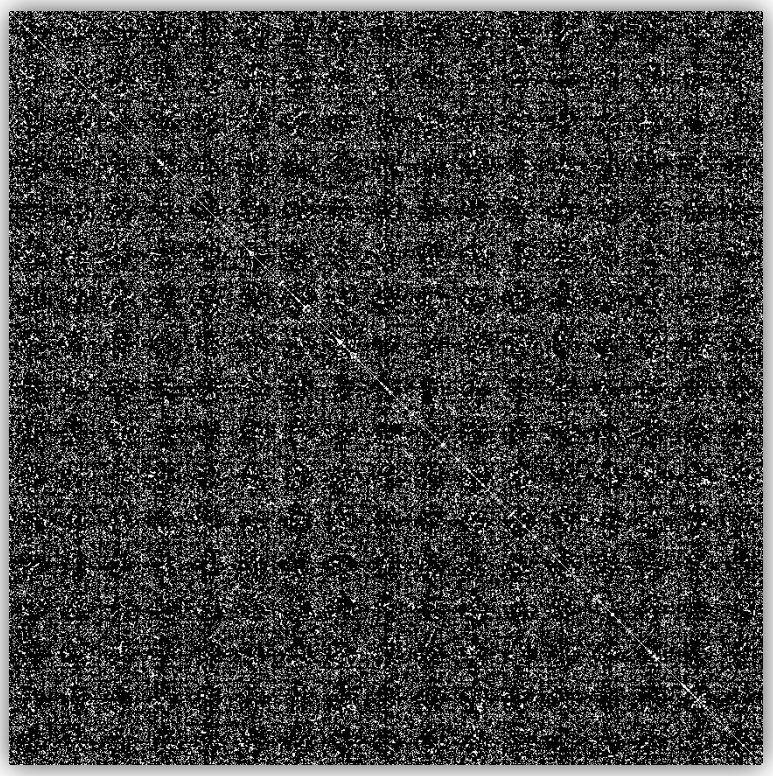


Fig. 30 Snapshot of the memory after the storage of 5000 images with correlation.

The weight matrix distribution confirms that the associative memory is acting as intended.

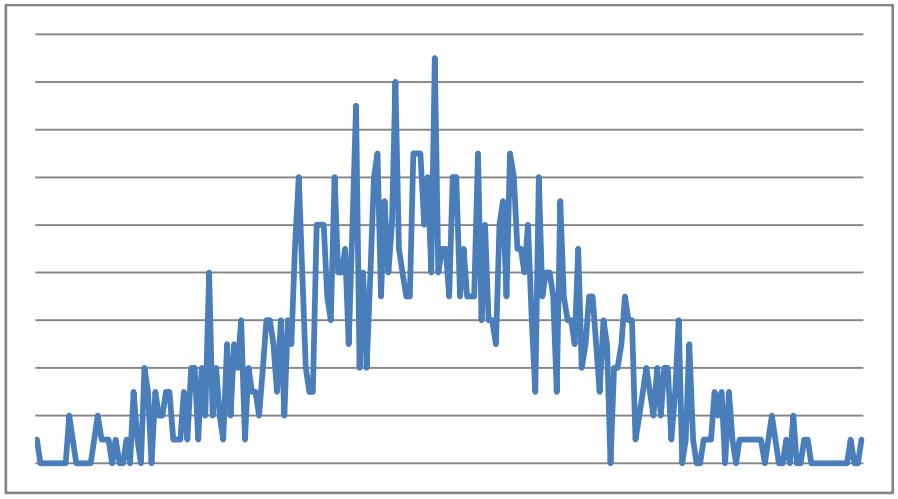


Fig. 31 Distribution of ones in the memory after the storage of 5000 images with correlation.

Ten thousand images

Figure 32 shows that the memory is quickly getting full. It can be anticipated that the forty thousand image memory diagram will present a memory filled with ones; which accounts for the results presented in Figure 28.

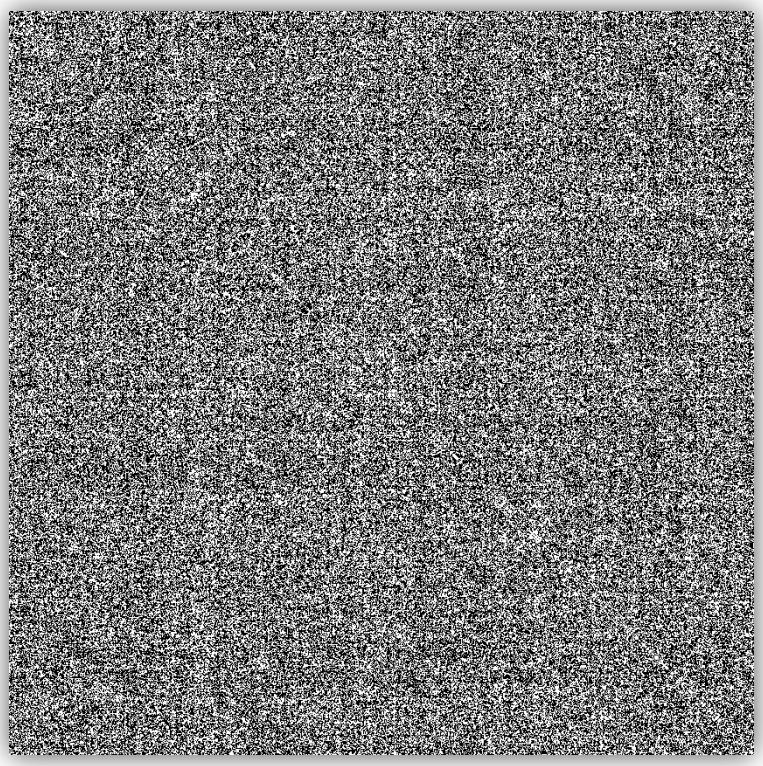


Fig. 32 Snapshot of the memory after the storage of 10000 images with correlation.

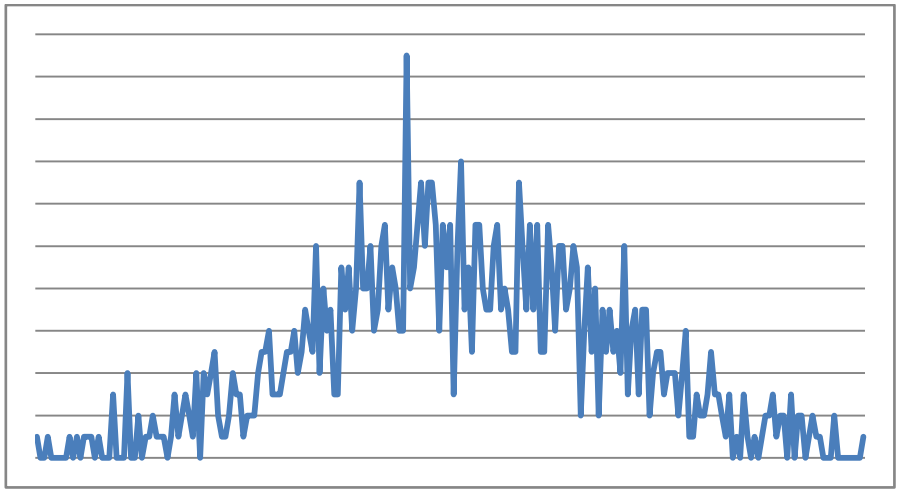


Fig. 33 Distribution of ones in the memory after the storage of 10000 images with correlation.

Forty thousand images

Figure 34 represents a full memory. From this snapshot the conclusion is that the data sets used in the experiment are too big. As the memory gets full, the tests will not be accurate because a filled memory will always return erroneous patterns in the recall phase.

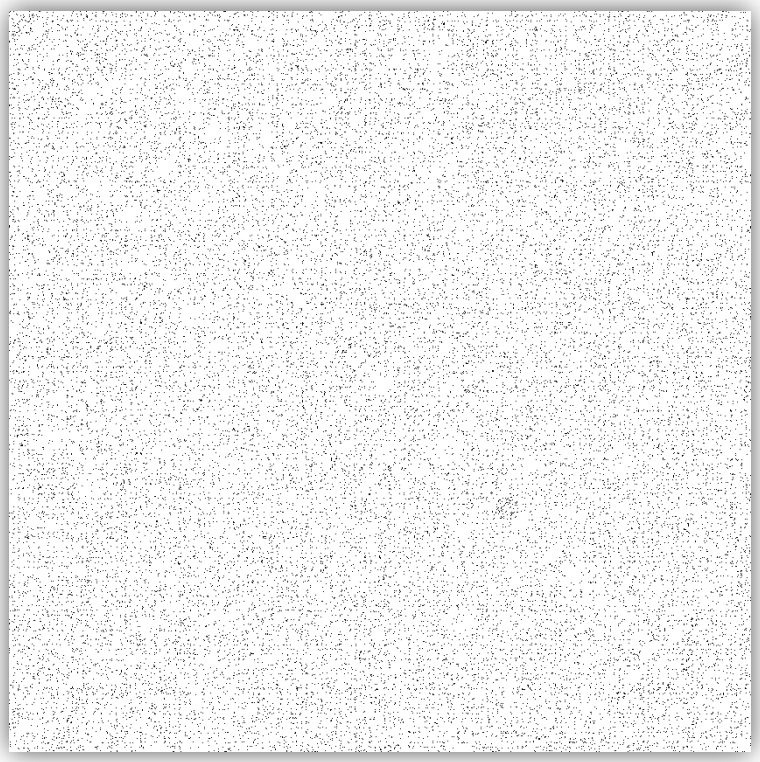


Fig. 34 Snapshot of the memory after the storage of 40000 images with correlation.

The weight matrix distribution indicates that the data is evenly distributed.

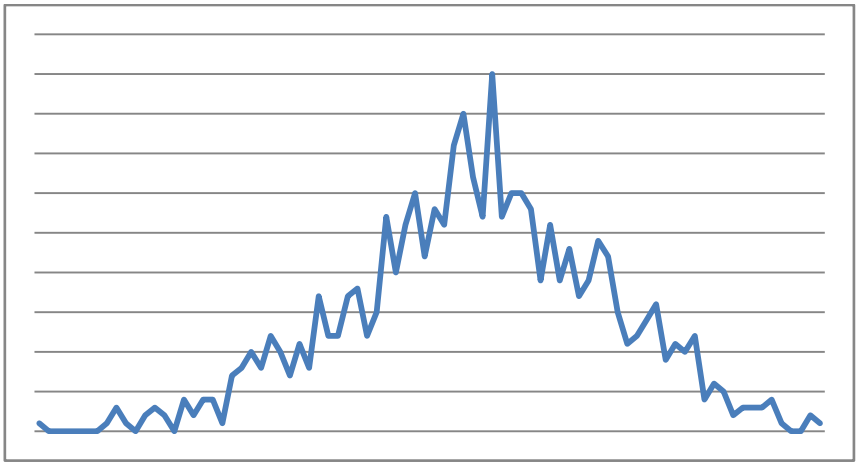


Fig. 35 Distribution of ones in the memory after the storage of 40000 images with correlation.

Aggregation coefficient 5

As we can see in the results below (see Figure 36) a higher aggregation will, as expected, find a similar pattern sooner than a smaller aggregation. With a higher aggregation the memory has a higher number of ones; the probability of finding a matched pattern is also higher.

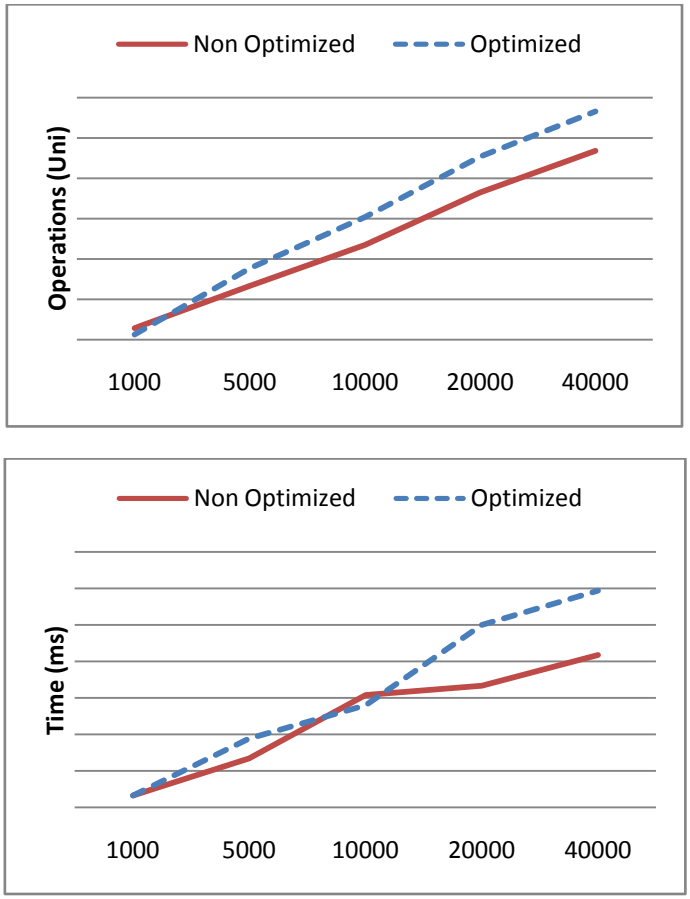


Fig. 36 Execution time and number of operations for Aggregation 5 (with correlation).

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.

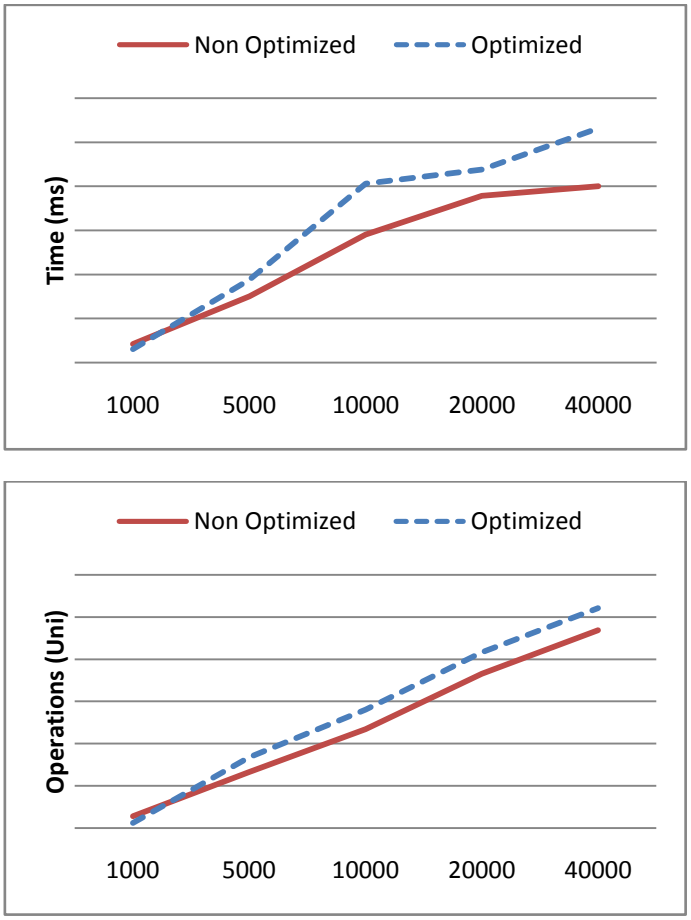


Fig. 37 Execution time and number of operations for Aggregation 10 (with correlation).

Three step aggregation (25, 10 and 5)

Having the previous experiments in mind, a good performance can't be expected from the three step aggregation process. If with an aggregation coefficient of 2 matched patterns were found with a small vector set, the three steps will only add overhead to the computation time and number of operations.

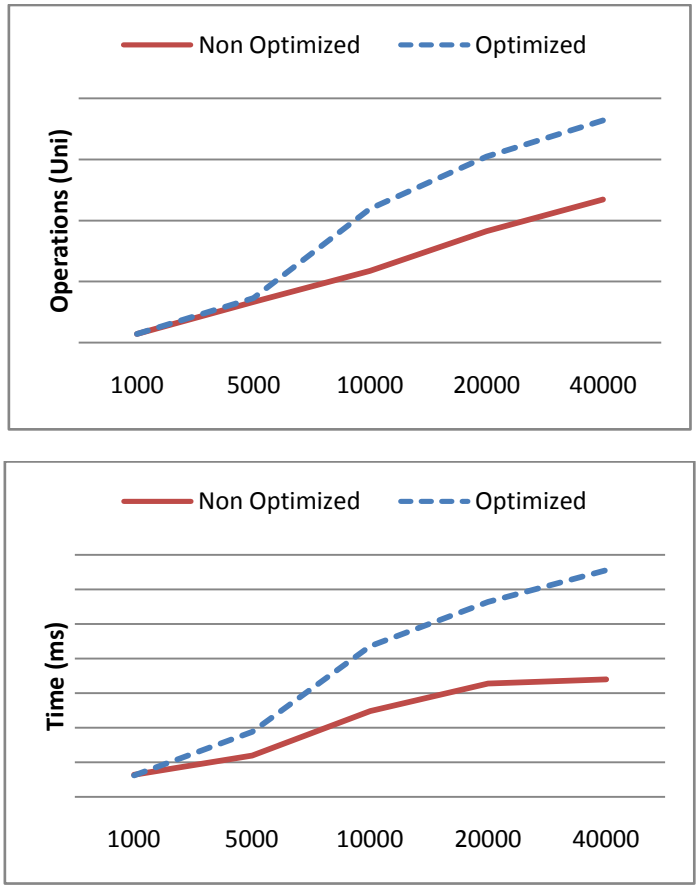


Fig. 38 Execution time and number of operations for three step aggregation process (with correlation).

Figure 38 indicates that, in the early data sets (1000 and 5000 vectors), depending on the input, only some of the steps are being executed. This explains the similar number of operations. In the larger data sets, with 40000 vectors, every step is processed because a matching pattern is always found.

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.

5.5 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 39), 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.

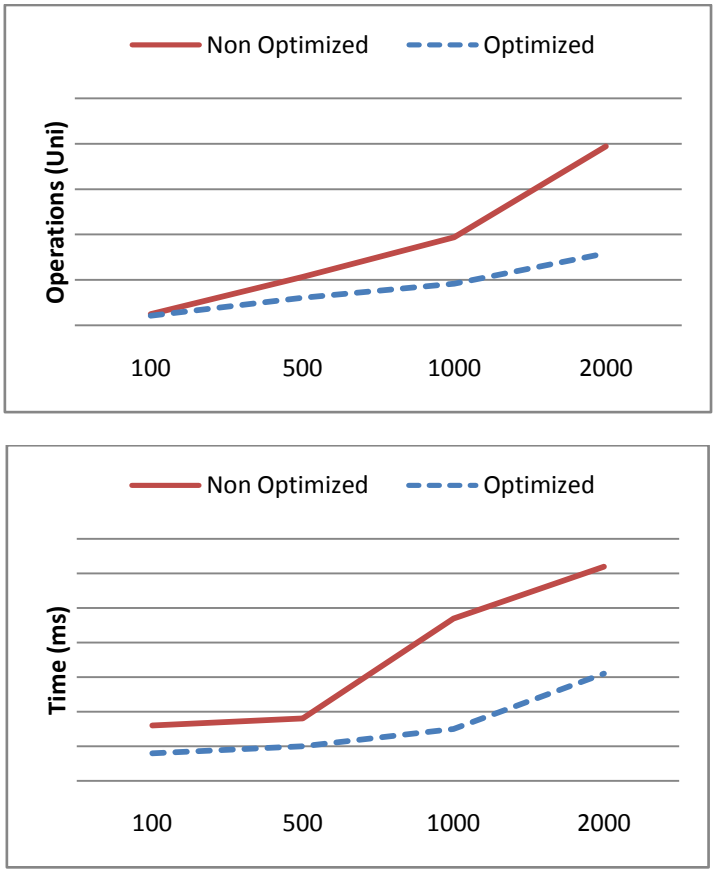


Fig. 39 Execution time and number of operations for Aggregation 2 (with correlation).

The following section will present the weight matrix diagram and distribution of the associative memory after storing 100, 500, 1000 and 2000 binary vectors with correlation.

5.6 Associative memory diagrams (correlation and smaller data sets)

One hundred images

The weight matrix diagram of the memory after storing one hundred images with correlation (Annex D.2) can be compared with the experiment of the non correlated one thousand stored patterns. Both images seem similar. The only difference is the diagonal which is characteristic of the correlation.

It is interesting to take a look at the distribution of ones in this experiment. The non correlated memory stores its data sequentially. As the data is randomly generated, the probability of having a vector without a single one is very low. The result is a homogeneous memory with its zeros and ones evenly distributed. In the other hand, with a smaller data set and correlation, the distribution will not be similar although the diagrams look the same. With correlation the achieved memory will have its ones concentrated, and many units will be full of zeros. Figure 40 confirms the statement above.

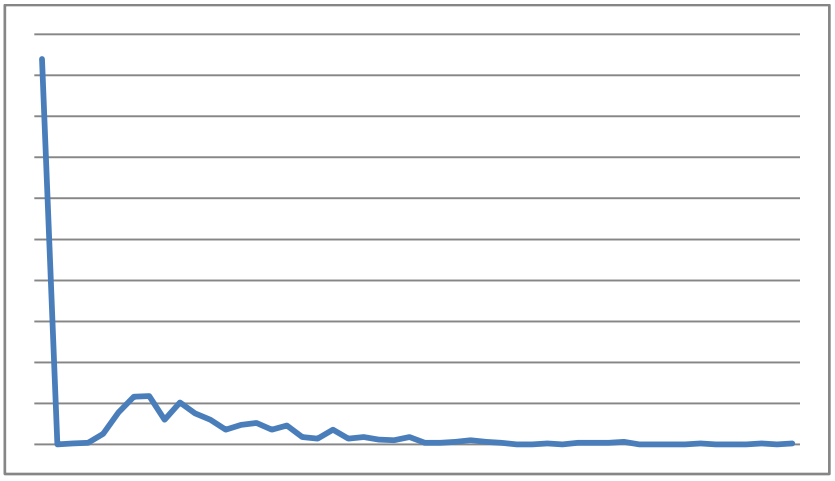


Fig. 40 Distribution of ones in the memory after the storage of 100 images with correlation.

Five hundred images

Annex D.3 illustrates a snapshot of the associative memory after 500 images were stored with correlation. Again, it can be confirmed that the data is random as the memory is homogeneous.

The distribution presented below (see Figure 41) indicates that, with 500 images, the memory is starting to behave as normal; meaning that, although there are some empty units, most of them are filled with ones.

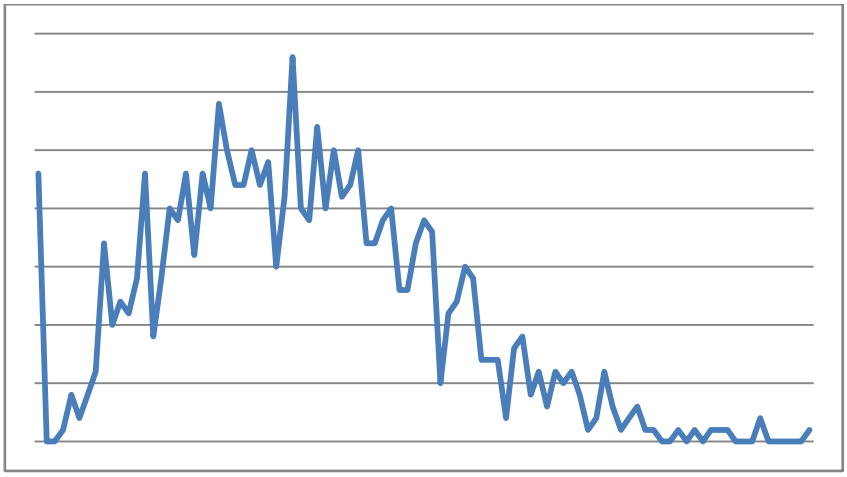


Fig. 41 Distribution of ones in the memory after the storage of 500 images with correlation.

The snapshot and distribution for the memory with 1000 images will not be presented here as it can be found in Figures 29 and Annex D.8.

Two thousand images

There is not much to say regarding the snapshot and distribution of the memory with 2000 stored images. Both present the usual and expected state.

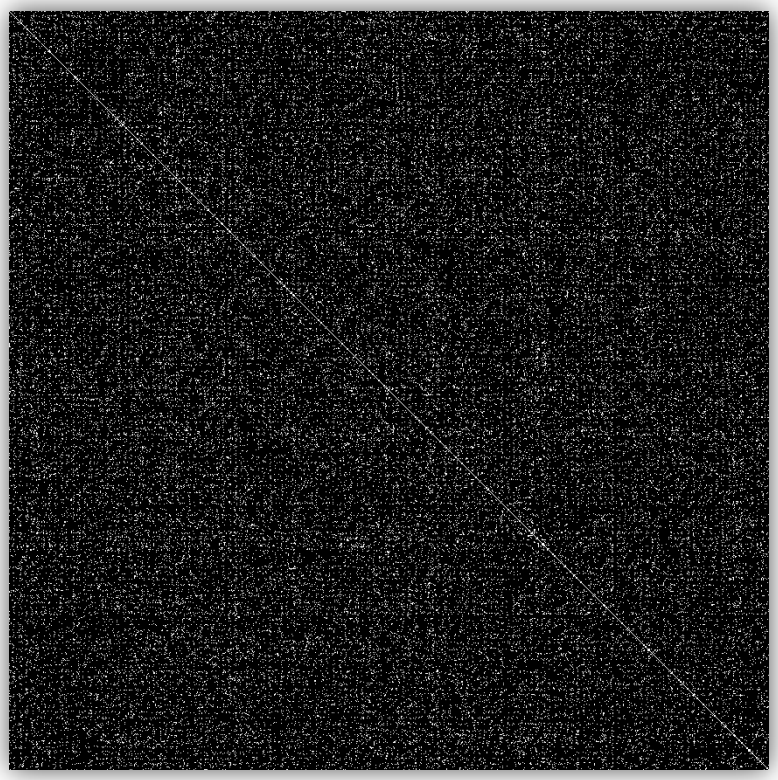


Fig. 42 Snapshot of the memory after the storage of 2000 images with correlation.

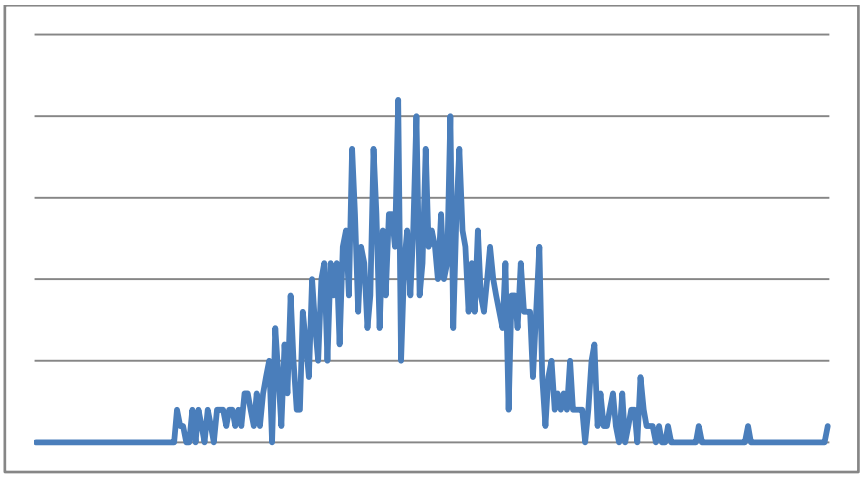


Fig. 43 Distribution of ones in the memory after storage of 2000 images with correlation.

Aggregation coefficient 5

As expected, the aggregation process for smaller data sets works quite well. Comparing this experiment with the coefficient 2, it can be seen a small decrease in the number of operations (see Figure 44).

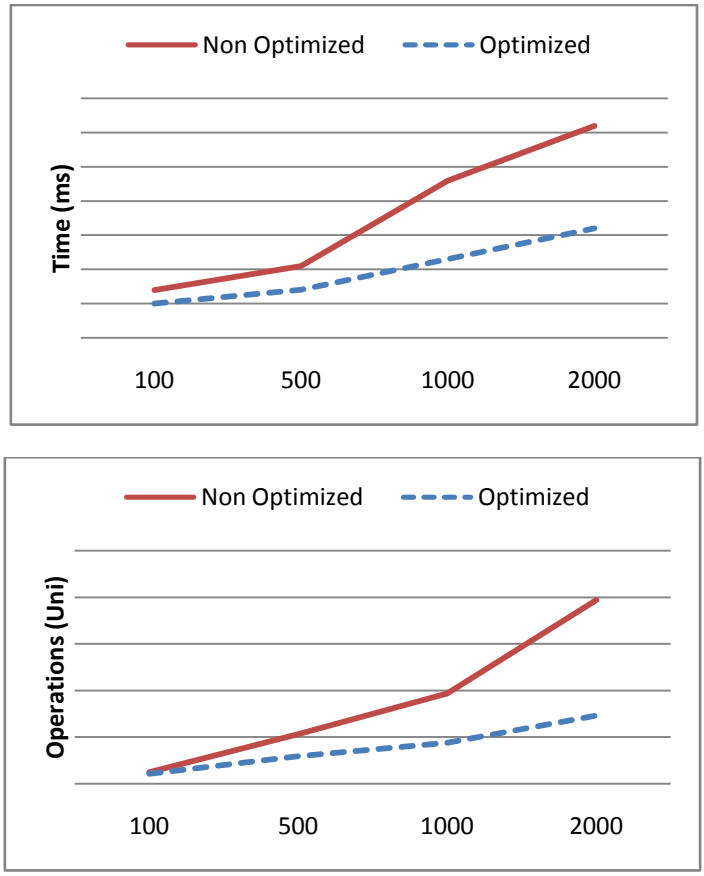
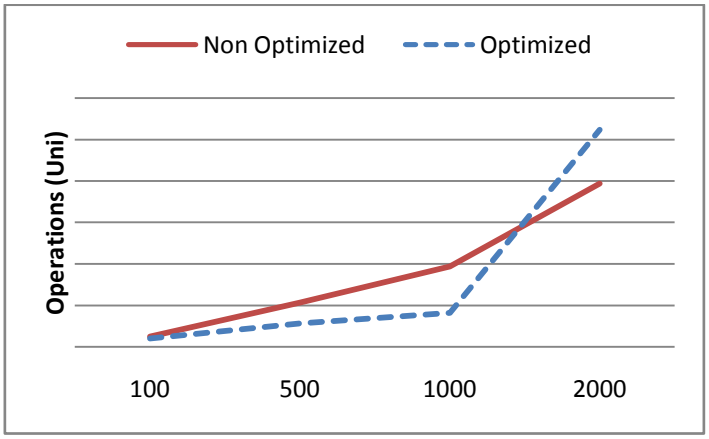


Fig. 44 Execution time and number of operations for Aggregation 5 (with correlation).

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 45).



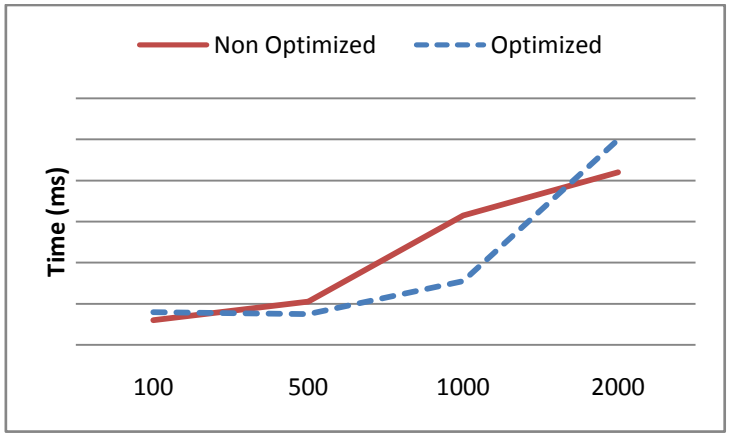
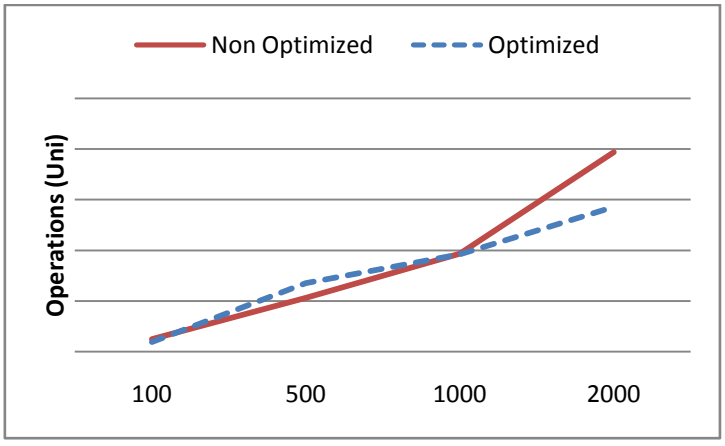


Fig. 45 Execution time and number of operations for Aggregation 10 (with correlation).

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.



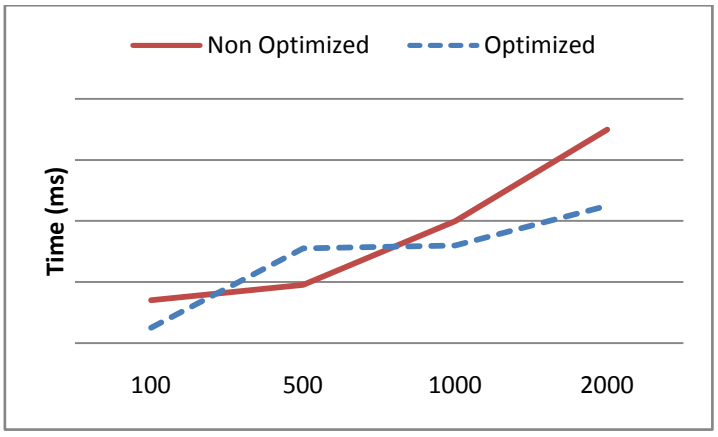


Fig. 46 Execution time and number of operations for three step aggregation method (with correlation).

5. 7 Conclusion

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. The results show that the ratio between the size of the vectors and number of stored images must be carefully selected. The fuller the memory is, the higher the probability of finding a matching pattern, in which case the aggregation step will only add to the execution time.

Also it is important to keep in mind that these are random generated data sets and therefore represent a worst case scenario. In real usage, data sets follow trends and will not occupy as easily the whole memory.

Chapter VI

Efficient Sparse Code results

This chapter 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 chapter:

- 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 first part of this section will illustrate, through diagrams and distribution of ones, why it is not possible to use common binary images with the *Lernmatrix*.

6.1 Common associative memory diagrams

Fifty percent ones

Annex D.9 represents the associative memory after the storage of 50, 100, 200 and 400 images. As can be seen, for every data set, the memory is completely full.

The distribution confirms that every single weight in the matrix is set to one.

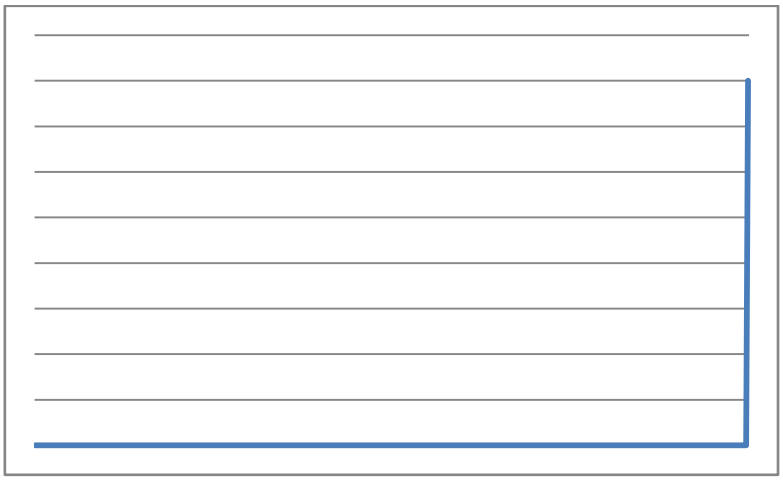


Fig. 47 Distribution of ones on the memory after the storage of 50, 100, 200 and 400 images with 50 percent ones.

Thirty percent ones (50 images)

With just thirty percent ones and a data set of fifty images, the diagram indicates that although the memory is not completely full, it is still overloaded and any query will result in an incorrect match (see Annex D.4).

The distribution is in accordance with the diagram depicted in Annex D.4.

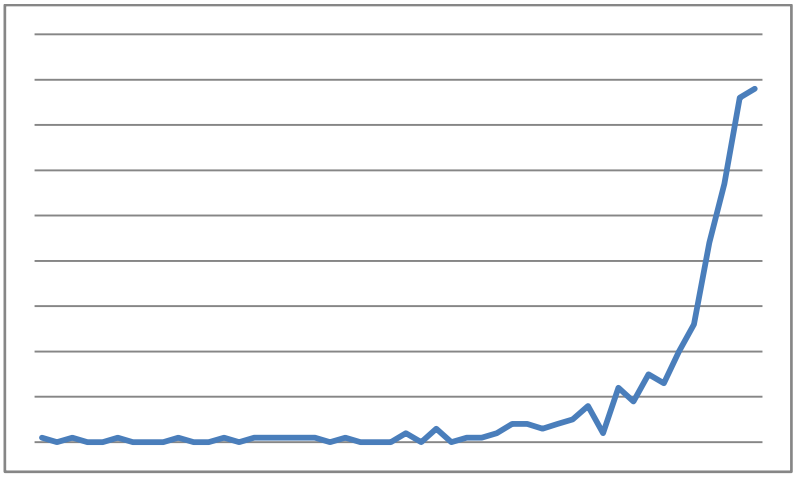


Fig. 48 Distribution of ones in the memory after storage of 50 images with 30 percent ones.

The diagrams and distributions regarding the 100, 200 and 400 data sets indicate a full memory similar to Figure 47 and Annex D.4 and therefore, will not be repeated.

Ten percent ones (50 images)

The results for 50 images with ten percent ones present a normal distribution and behavior (see Figure 49). Unfortunately most binary images contain more than ten percent ones, which mean that this data does not represent a real case scenario.

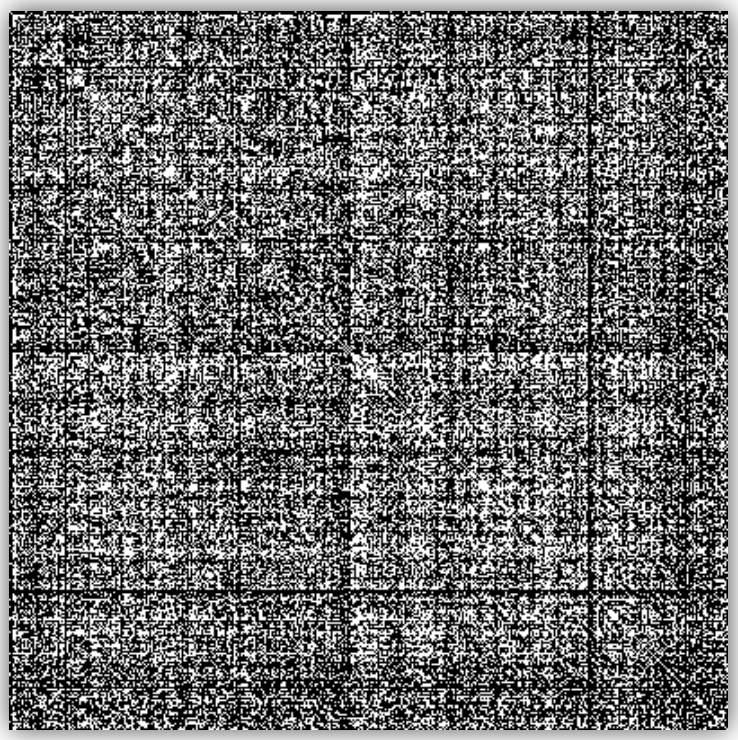


Fig. 49 Snapshot of the memory after the storage of 50 images with 10 percent ones.

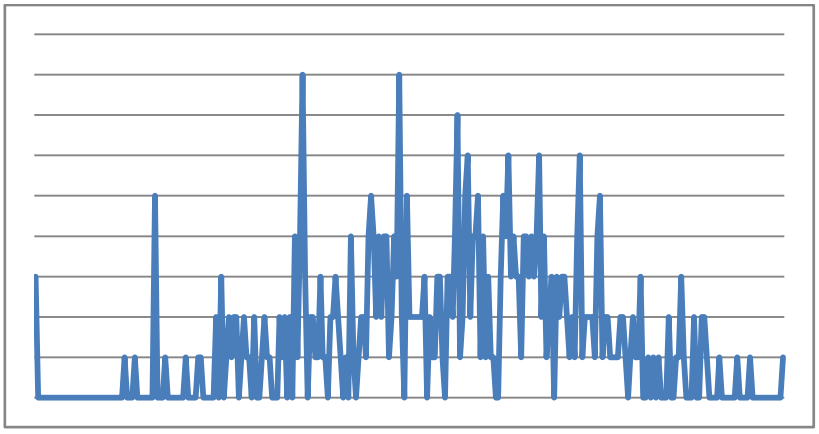


Fig. 50 Distribution of ones on the memory after storage of 50 images with 10 percent ones.

As can be seen from the collected information on this chapter, data must be sparse in order to be used in the *Lernmatrix*. The scenario presented in Annex D.9 and Figure 47 confirms why it is not possible to use non sparse data with the associative memory. An overloaded memory will always recall incorrect patterns.

6.2 First set of tests (data sets of 50, 100, 200 and 400 images)

The results provided in this chapter consist of the tests done with images containing 50, 30, 20, and 10 percent ones. This indicates that, for instance, an image with a size of 400 (20x20) and 10 percent ones will have in average forty weights set to one distributed randomly across its length. 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 51 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.

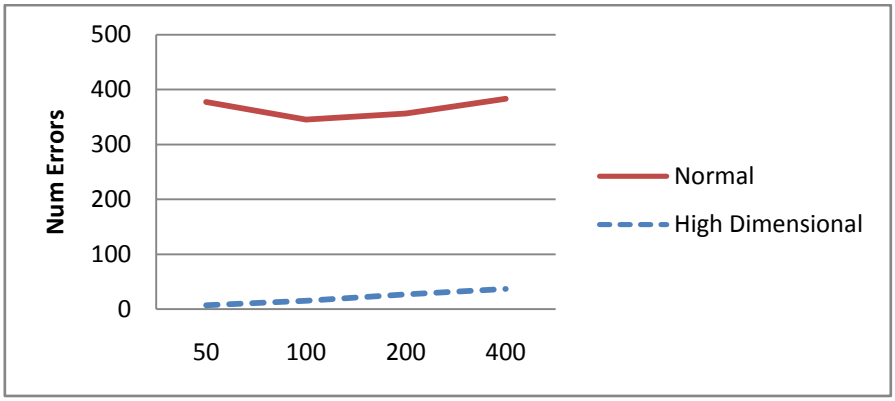


Fig. 51 Number of errors in both the high dimensional and *normal* associative memories for 50 percent ones.

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 52). As was illustrated in Figure 51, the associative memory with high

dimensional space presents a lower number of errors that is steadily increasing as the number of stored patterns grows.

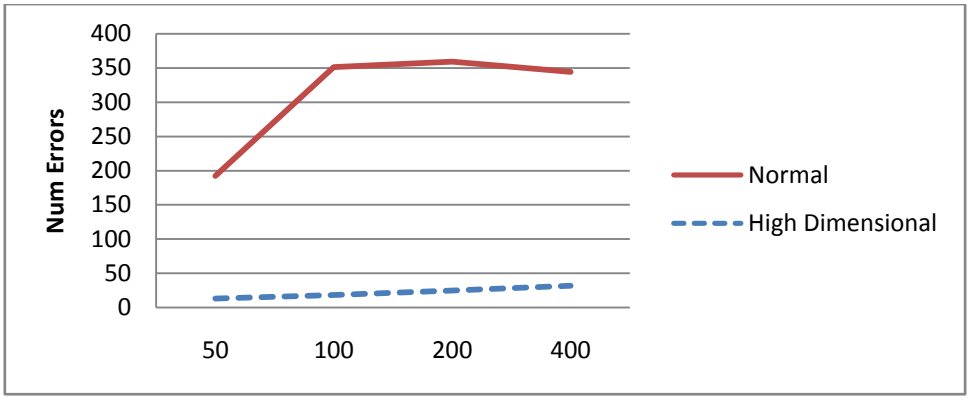


Fig. 52 Number of errors in both the high dimensional and *normal* associative memories for 30 percent ones.

Twenty percent ones

As expected, because the number of ones in the images is decreasing, the associative memory is becoming sparser. Figure 53 indicates that there is a linear correspondence between the number of stored images and the errors on the *normal* memory.

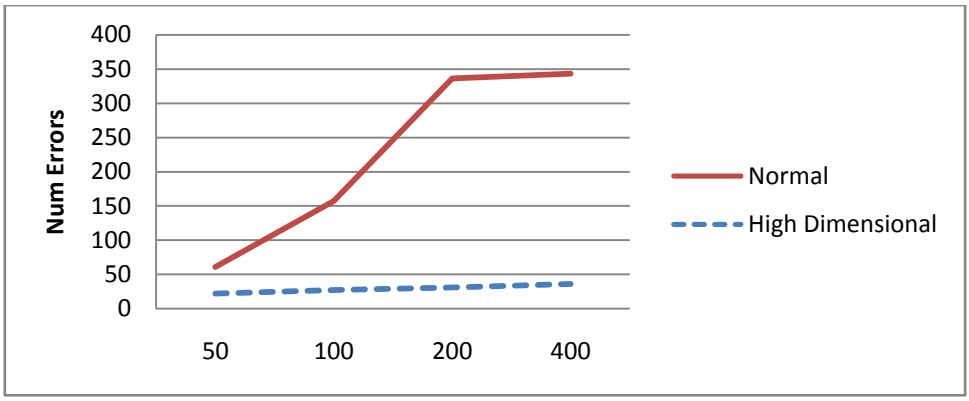


Fig. 53 Number of errors in both the high dimensional and *normal* associative memories for 20 percent ones.

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 54).

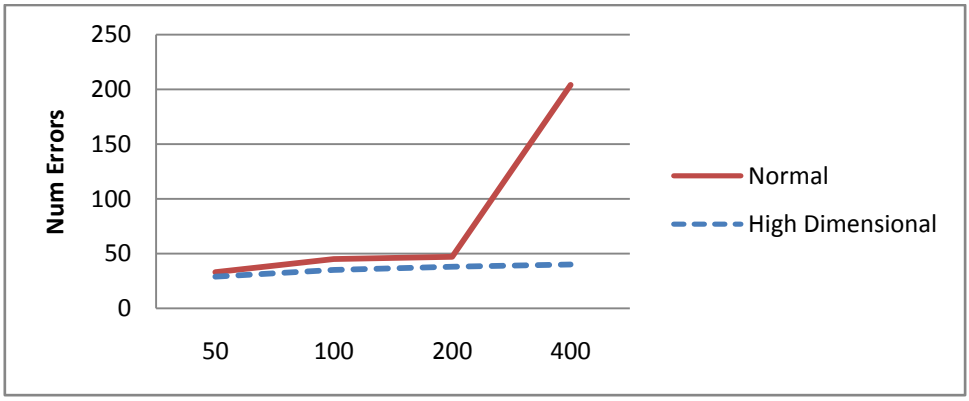


Fig. 54 Number of errors in both the high dimensional and *normal* associative memories for 10 percent ones.

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

6.3 Second set of tests (data sets of 400, 800, 1200 and 1600 images)

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 55).

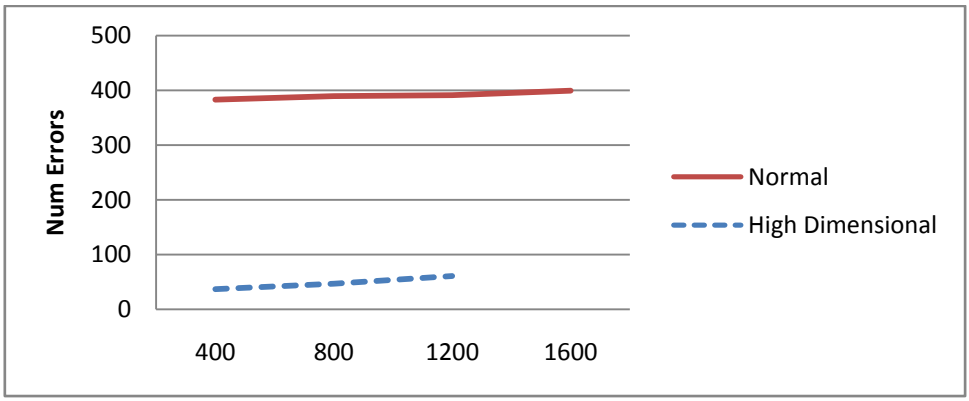


Fig. 55 Number of errors in both the high dimensional and *normal* associative memories for 50 percent ones.

The next section will present the weight matrix diagram and weight matrix distribution of the higher dimensional memory.

6.4 Associative memory with high dimensional space diagrams

Technical difficulties prevented from acquiring the weight matrix diagram and distribution for the high dimensional memory from the previous tests. In order to have an understanding of its state another smaller sized matrix was produced.

Eight hundred images

Annex D.5 indicates that when the coding of the original matrix is mapped into the high dimensional space there is some space near the diagonal that is not used. Coincidently the sum of the *squares* near the diagonal is exactly the number of the window that was chosen to

code the original matrix (eight). Further information can be obtained through the distribution diagram.

The distribution diagram shows a peculiar behavior and confirms the diagram depicted in Annex D.5. There is some space of the memory that is not being used.

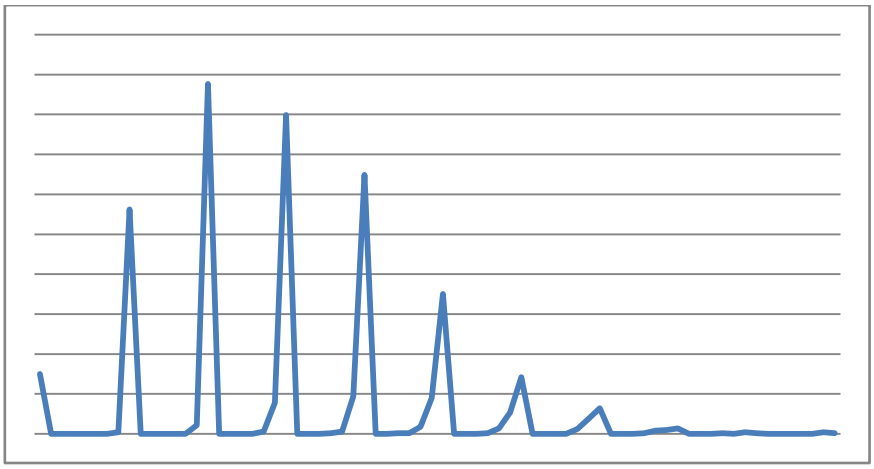


Fig. 56 Distribution of ones in the high dimensional memory after storing 800 images with 50 percent ones.

Sixty four images

In order to maintain the same proportion between the number of images and unit size, it was decided to reduce the number of images from the previous experiment. This will present an analogous example to the one with 400 images and a unit size of 400. Looking carefully at the weight matrix diagram from Annex D.6 and comparing it with the previous diagram (see Annex D.5), it can be concluded that both present similar behaviors although the current diagram illustrates a sparser memory.

The distribution also confirms the similarity of behavior between the 64 and 800 data set experiments (see Figure 57).

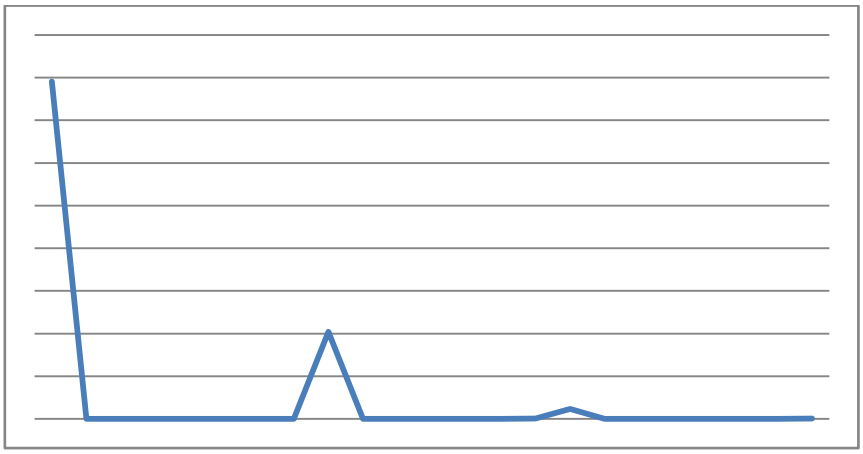


Fig. 57 Distribution of ones in the high dimensional memory after storing 64 images with 50 percent ones.

After some consideration it is possible to explain the behavior illustrated in the previous tests. Consider the following input patterns:

- 0010;

- 1101
- 1010;
- 0001;

For a window of size 2, the unary representation of these vectors is:

- 0 2;
- 3 1;
- 2 2;
- 0 1;

The correspondent binary translation would be (considering that the zero is represented by a clear vector for simplification purposes):

- 0000 0100;
- 1000 0010;
- 0100 0100;
- 0000 1000;

According to the formulations presented in chapter IV, the size of the high dimensional space is:

$$2^l \times \frac{n}{l} \leftrightarrow 2^2 \times \frac{4}{2} = 8$$

After storing these patterns in an empty memory we would get the matrix represented in Figure 58:

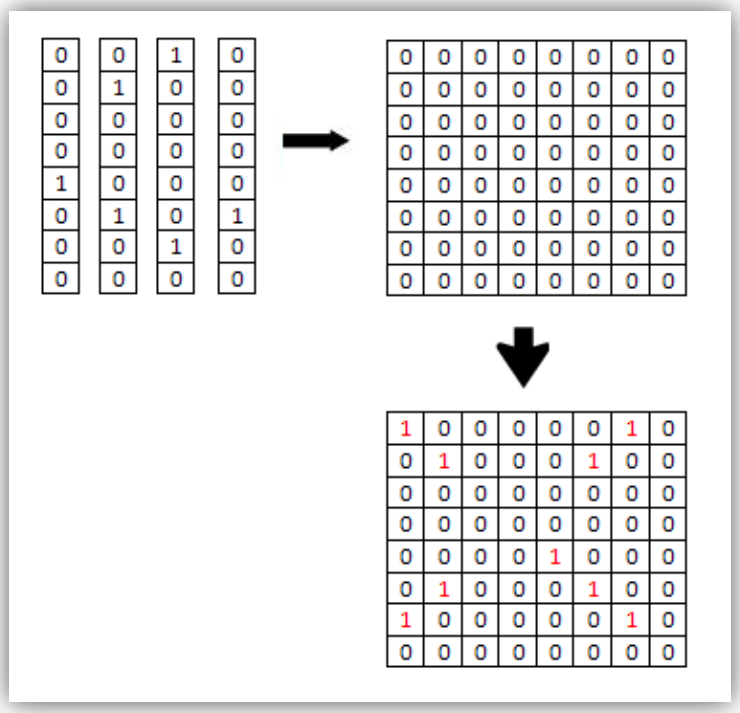


Fig. 58 Associative Memory with high dimensional space after the storage of several patterns.

The results indicate that, similar to what happened in the previous experiments, the space around the diagonal is unused. This can be explained due to the nature of the unary coding combined with the correlation. As only one bit is set to one after each translation, the entire space around that bit will be empty.

6.5 Conclusion

The experiment results demonstrate the potential of the coding method. The number of errors after the recall of a non sparse pattern in the high dimensional space is several times lower than the one in the *normal* one. Although it was not possible determine its limits, the usage of this method will make it viable to use non sparse data on the *Lernmatrix*. It is important to mention that in the process of the experiments it was found out that a small portion of the high dimensional space is not being used.

As an example of a possible formulation for the discovery of the optimal coding parameters, consider the *Lernmatrix M* calculation:

$$(1)\ M = \log_2\left(\frac{n}{4}\right) \leftrightarrow n = 2^M \times 4$$

For an image with a size of *n* it is assumed that half its content is black; in other words the vector will be half filled with ones. Assuming that each one is distributed across the length of the vector it is safe to assume that for whatever chosen window size *l*, the number of ones in the high dimensional space *B*, is $^{n}/_{l}$. The high dimensional space *U* size is:

$$(2)\ U = \frac{n}{l} \times 2^l$$

The optimal coding values are those for which *U* and *B* are estimated to be closer to *n* and *M* respectively (see (1)). For instance, consider a vector with a size 100. If the window is 8 then:

$$U = \frac{100}{8} \times 2^8 \leftrightarrow U = 3200\ and\ B = \frac{100}{8} \leftrightarrow B \cong 13$$

According to (1) for *M* = 13, n should be 32768. For a window size of 9 then:

$$U = \frac{100}{9} \times 2^9 \leftrightarrow U \cong 5700\ and\ B = \frac{100}{9} \leftrightarrow B \cong 11$$

The optimal value for *M* = 11 is *n* = 8192. It can be concluded that a window size of 9 is better than 8.

Chapter VII

Conclusion

The principal objective of this thesis 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 chapter IV. Through the results illustrated in chapter V and VI 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 39). 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 chapter VI 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 space of the associative memory 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.

7.1 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 the 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 efficient sparse code method is an opportunity to test and potentiate its application.

7.2 Personal Contributions

This project was a good opportunity to experience research work. This kind of research is both more demanding and rewarding if you compare it with most of the projects where expectations are clearly defined since the beginning. The ups and downs and sometimes lack of comprehension made this work a bittersweet experience. It is undeniable, though, that due to the different fields of expertise that were put to use in order to complete the project, I find myself a more competent and enriched person. It was also important for me to understand how the previously acquired knowledge could be put to use in the name of science.

It is also a joy to know that everything that was done will be continued and that this project may be used to improve databases in the future.

Section II

Bibliography

Bibliographic References

1. **P. Churchland, T. Sejnowski.** *The Computational Brain.* s.l. : The MIT Press, 1994.

2. **L. Squire, E. Kandel.** *Memory: From Mind to Molecules.* s.l. : W. H. Freeman, 2000.

3. **M. Glesner, W. Pochmuller.** *Neurocomputers: An Overview of Neural Networks in Vlsi .* s.l. : Chapman & Hall, 1994.

4. **H. Simon, A. Newell.** *Human Problem Solving.* s.l. : Prentice Hall, 1972.

5. **Hassoun, M.** *Associative Neural Memories: Theory and Implementation.* s.l. : Oxford University Press, 1993.

6. **Churchland, P.** *The Engine of Reason, the Seat of the Soul.* s.l. : The MIT Press, 1996.

7. **Anderson, J.** *An Introduction to Neural Networks.* s.l. : The MIT Press , 1995.

8. **J. Hertz, A.Krogh, R. Palmer.** *Introduction to the Theory of Neural Computation.* s.l. : Westview Press, 1991.

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

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

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

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

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

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

15. **Denning, Peter J.** *Sparse Distributed Memory.* s.l. : American Scientist, July, 1989.

16. **Wichert, A.** *Associative Computation, Phd Thesis.* 2000.

17. **Wichert, A.** *A categorical expert system: Jurassic.* 2000.

18. **Wichert, A.** *Associative computer: a hybrid connectionistic production system.* 2005.

19. **Feldman, J.** *Four frames suffice: A provisional model of vision and space.* s.l. : Behavioral and Brain Sciences, Vol. 8, pp. 265-289, 1985.

20. **Hebb, D.** *The Organization of Behavior: A Neuropsychological Theory .* s.l. : Lawrence Erlbaum, 1949.

21. **T. Sejnowski, G. Tesauro.** *The Hebb Rule for Synaptic Plasticity: Algorithms and Implementations.* s.l. : NY: Academic Press, 1989.

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

23. **Samarasinghe, S.** *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition.* s.l. : Auerbach Publications, 2006.

24. **Hopfield, J.** *"Neural" computation of decisions in optimization problems .* s.l. : Springer Berlin / Heidelberg, 1985.

25. **Hopfield, J.** *Neural Networks and Physical Systems with emergent collective computational abilities.* s.l. : Proc. NatL Acad. Sci. USA Vol. 79, pp. 2554-2558, 1982.

26. **DeLiang Wang, Joachim Buhmann.** *Pattern Segmentation in Associative Memory.* s.l. : The MIT Press, 1990.

27. **Kanerva, P.** *Sparse Distributed Memory and related models.* s.l. : The MIT Press, 1993.

28. **U. Ramamurthy, S. D'Mello, S. Franklin.** *Realizing Forgetting in a Modified Sparse Distributed Memory System.* s.l. : Proceedings of the 28th Annual Meeting of the Cognitive Science Society, 2006.

29. **Hamming, R.** *Error-detecting and Error-correcting Codes.* s.l. : The Bell System Technical Journal, Vol 2, 1950.

30. **Keeler, J.** *Comparison between Kanerva's SDM and Hopfield-type neural networks.* s.l. : Cognitive Science 12:299–329, 1988.

31. **Kosko, B.** *Bidirectional Associative Memories.* s.l. : IEEE Transactions on Systems, Man, and Cybernetics, Vol 18, 1988.

32. **Sarkar, D.** *An Algorithm for Computation of Inter-Pattern Interference noise in BAM.* s.l. : Neural Network World, Vol. 12, No. 1, pp. 67-73, 2002.

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

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

35. **Lapir, G.** *Associative Memory.* s.l. : IEEE, vol. 1, pp.: 531-536, 1998.

36. **Tversky, A.** *Feature of similarity.* s.l. : Psychological Review, 84:327-352, 1977.

37. **Bai-ling Zhang, Y. Miao, G. Gupta.** *Associative Memory Model for face recognition.* s.l. : Springer Berlin, 2005.

38. **A. Maurer, M. Hersch, A. Billard.** *Extended Hopfield Network for Sequence Learning: Application to Gesture Recognition.* s.l. : Presented at ICANN'2005, 2005.

39. **Stevens, K.** *Acoustic Phonetics.* s.l. : MIT Press, 2000.

40. **C. Williams, K.Stevens.** *Emotions and Speech: Some Acoustical Correlates.* s.l. : Journal of the Acoustical Society of America, 52, 4, 1238-1250, 72, 1972.

41. **Al-Shoshan, A.** *Speech and Music Classification and Separation: A Review.* s.l. : J. King Saud Univ., Vol. 19, Eng. Sci. (1), pp. 95-133, 2006.

42. **J. Minghu, L. Biqin, Y. Baozong.** *Speech Recognition by using the extended associative memory.* s.l. : Intelligent Processing Systems, ICIPS '97, 1997.

43. **G. Borden, K. Harris, L. Raphael.** *Speech Science Primer: Physiology, Acoustics, and Perception of Speech.* s.l. : Lippincott Williams & Wilkins, 2002.

44. **P. Herrera-Boyer, G. Peeters, S. Dubnov.** *Automatic Classification of Musical Instrument Sounds.* s.l. : Audiovisual Institute - Pompeu Fabra University, 2006.

45. **G. Munjal, S. Kaur.** *Comparative study of ANN for Pattern Classification.* s.l. : Int. Conf. on Mathematical Methods and Computational Techniques in Electrical Engineering, Bucharest, October 16-17, 2006.

46. **Shannon, C.** *Prediction and entropy of printed English.* s.l. : The Bell System Technical Journal, 30:50-64, 1951.

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

48. **Goldstone, R.** *Similarity.* s.l. : MIT Encyclopedia of the Cognitive Sciences, 1999.

49. **E. Smith, S. Sloman.** *Similarity- versus rule-based categorization.* s.l. : em Cognit, Vol. 22, No. 4, pp. 377-386., 1994.

50. **Wichert, A.** *Pictorial Reasoning with Cell Assemblies.* s.l. : Connection Science, Volume 13, Number 1, pp. 1-42, 2001.

51. **F. Garfias, J. Leon, C. Marquez.** *Steinbuch's Lernmatrix: Theoretical Advances.* s.l. : Computación y Sistemas. Vol. 7 No. 3 pp. 175 - 189, 2004.

52. **H. Bosch, F. Kurfess.** *Information storage capacity of incompletely connected associative memories.* s.l. : Elsevier Science Ltd, 1998.

53. **J. Buhmann, DeLiang Wang.** *Pattern Segmentation in Associative Memory.* s.l. : The MIT Press, 1990.

54. **Oherson, D.** *New axioms for the constrast model of similarity.* s.l. : Journal of Mathematical Psychology, Volume 31 , Issue 1 , 1987.

55. **Biederman, I.** *Surface vs edge-based determinants of visual recognition.* s.l. : Cognitive Psychology, Vol 20 Issue 1, 1988.

56. **Osherson, D.** *Probability judgement.* s.l. : The MIT Press, 1995.

Section III

Appendices

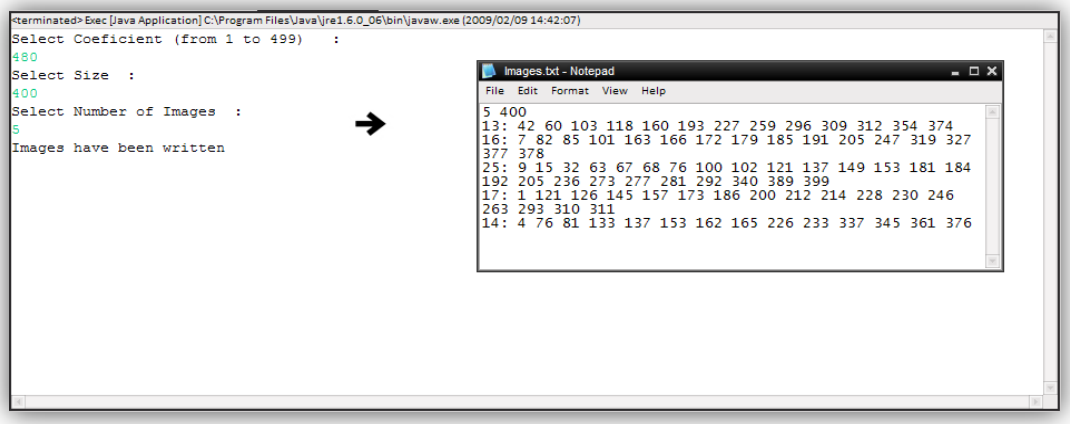
Appendix A

Users Manual

As was presented in chapter I, there were two major objectives to this project. The first was to reduce the query time on Steinbuch's *Lernmatrix*, and the second to allow the usage of images, or non sparse data, in an associative memory. Both objectives were resolved in different applications. These applications will be demonstrated in the following paragraphs.

Random Image Generator

Before the associative memory was implemented it was important to have a way to generate the data that was to be used on the testing phase. In order to produce this data, an application was developed that permits the creation of random vectors in pointer representation. The figure below demonstrates an example of the data generator.



Coefficient: The coefficient is the probability factor that controls the sparseness of the vector. In the example above, a coefficient of 480 will determine an average of 16 ones in a vector of a size 400.

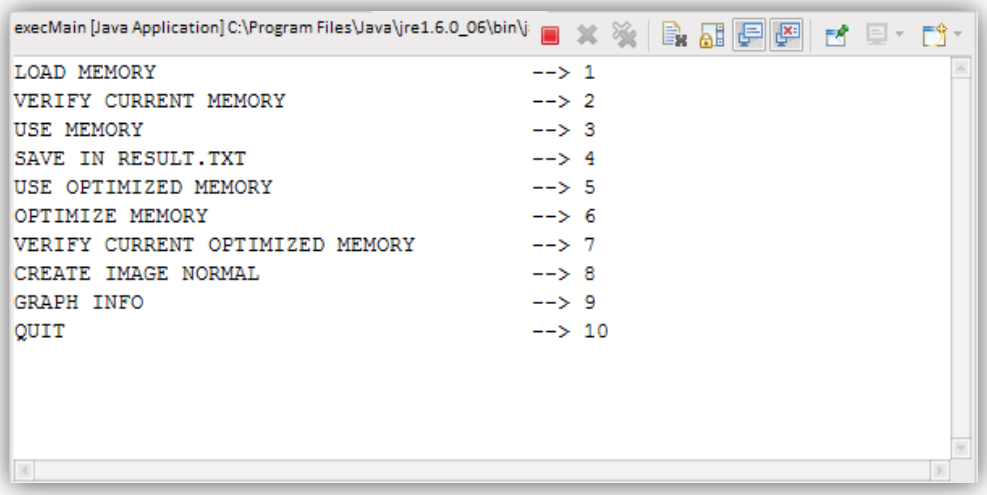
Size: This option allows the user to select the size of the vector. As can be seen in the figure, it was selected a size of 400.

Number of images: This parameter allows the user to choose how many images will be generated.

As was shown in chapter IV, the pattern vectors are represented by the number of ones and their positions in the vector. The output of the random image generator is a file containing in the first line the number of images and size followed by each image.

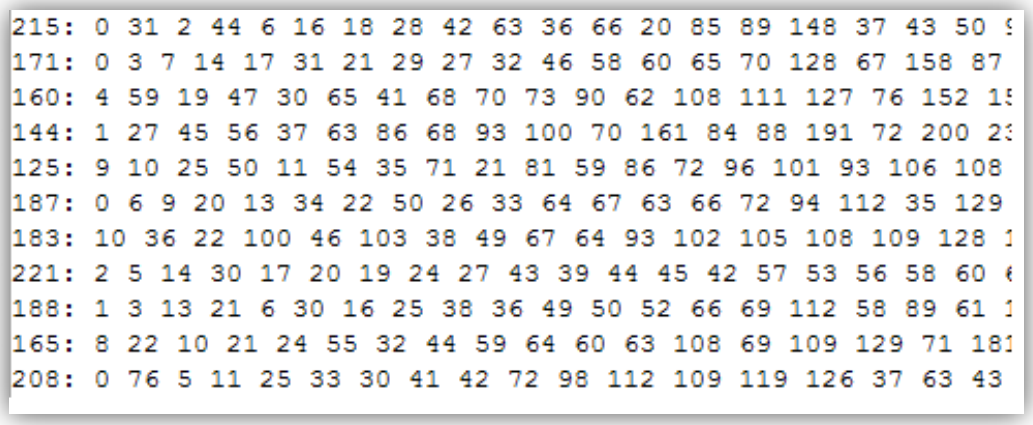
Hierarchical Associative Memory

Consider the figure below that represents a console with a list of all the options permitted by the *speedup* application. Each of these options will be described further below.



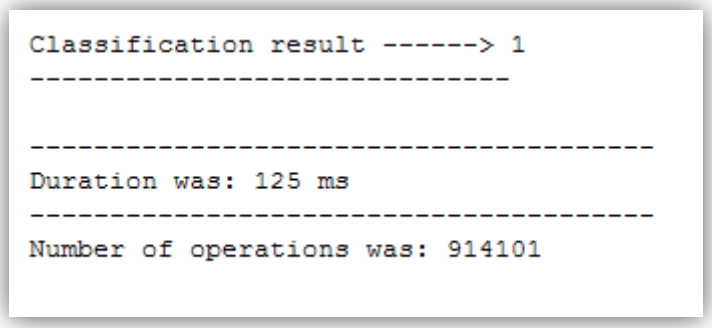
Load Memory: The storage of patterns is permitted through this option. The patterns must be represented in pointer representation as depicted in chapter IV. There is no significant return parameter.

Verify Current Memory: This option will allow to verify the state of the associative memory. The figure below illustrates an example of a memory after the storage of 1000 patterns.



Each line is a unit of the memory.

Use Memory: In order to query the memory, the use memory option is used. This allows for a pattern to be presented to the memory and for the most similar pattern to be recalled.



The classification result indicates whether a matching pattern was found or not. A one represents a *hit* and a zero a *miss*. The duration time and number of operations are self explanatory parameters and represent the execution time of the query and number of operations needed.

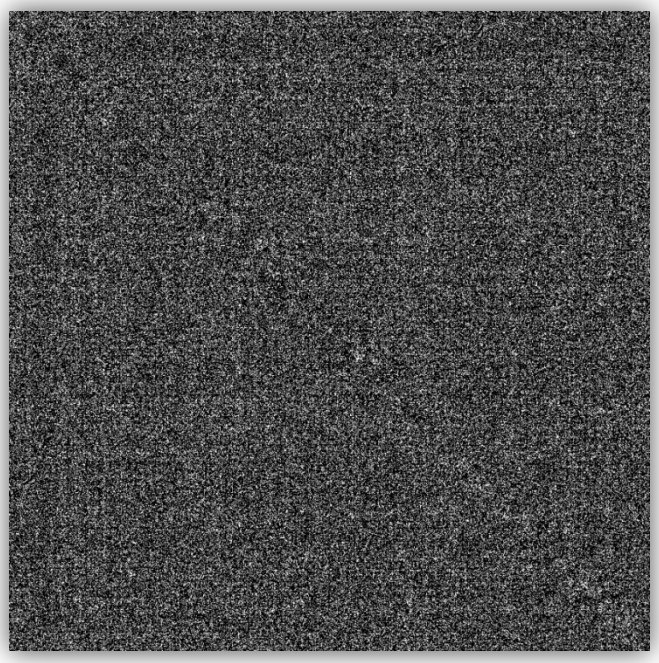
Save in result.txt: Executing this method will save the associative memory in a text file called result.txt.

Optimize Memory: The optimize memory option creates a secondary associative memory, calculated from the first, according to the previously defined aggregation parameter. This method has no return value.

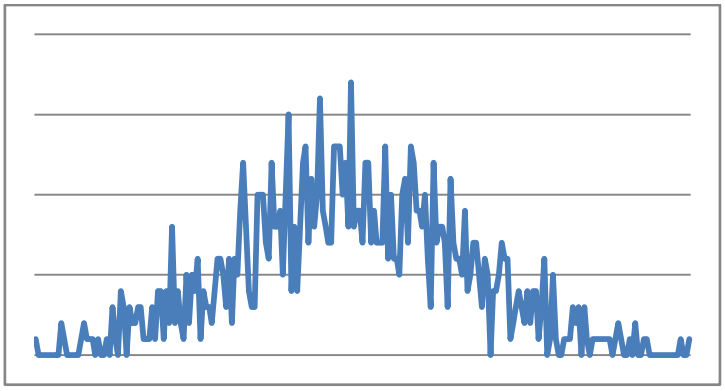
Use Optimized Memory: Similar to the use memory option, the use optimized memory method calculates the most similar pattern in the aggregated memory. The result is the same as illustrated in the figure above.

Verify Current Optimized Memory: Prints the aggregated memory in order to verify its state.

Create Image Normal: Creates a weight matrix diagram of the memory, similar to the one presented below.



Graph Info: This option allows for the computation of the weight matrix distribution; also illustrated below.

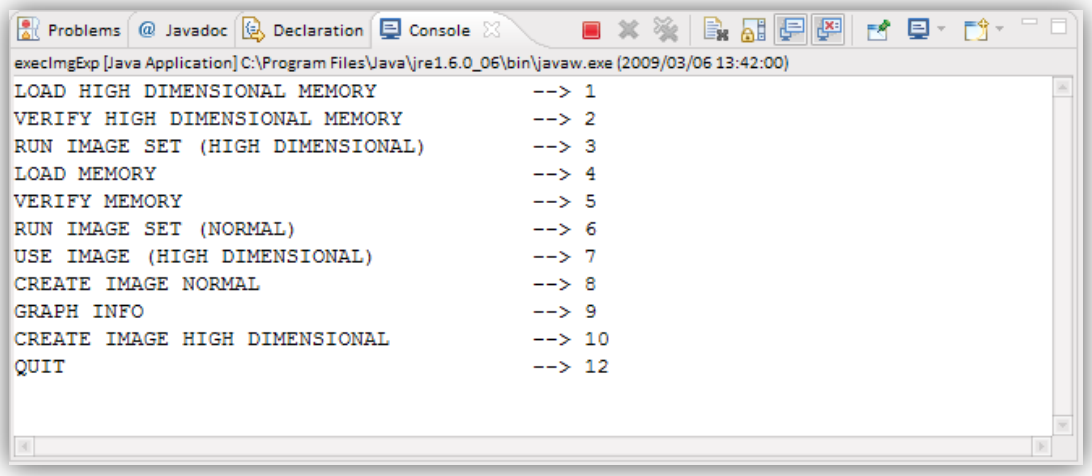


Example

The initial phase of a typical test sequence would be to load the memory and recall a pattern from it. After registering the results (number of operations and time), the memory would be aggregated and the same pattern would try to be recalled from it. The results are then compared, interpreted and consolidated in a table.

Efficient sparse code

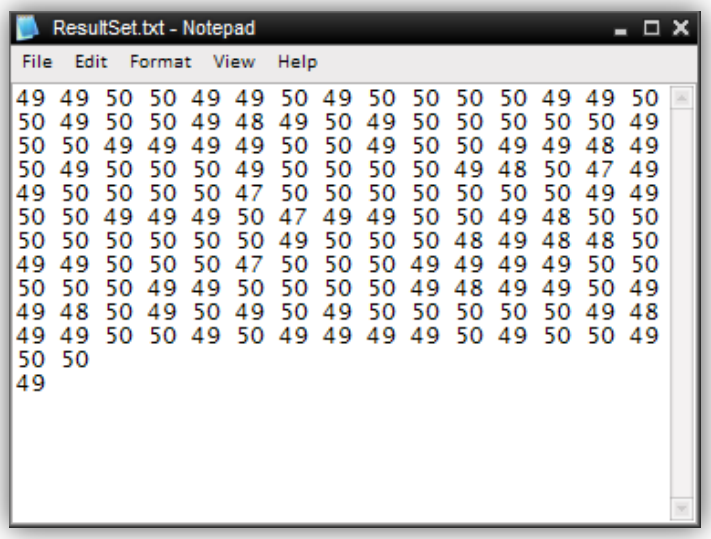
The figure below illustrates the high dimensional space memory application.



Load High Dimensional Memory: The creation and subsequent storage of patterns in the high dimensional memory is done through this option. The patterns are coded in the sparse representation and then stored in the memory with correlation.

Verify High Dimensional Memory: This option will allow to verify the current units of the high dimensional associative memory.

Run Image Set (High Dimensional): The execution of this method allows to recall all the patterns depicted in a given text file. This option is used to thoroughly test all the patterns that are stored in the memory. The result is a text file named ResultSet.txt similar to the one illustrated below.



Each number represents the number of errors returned in each recall phase. The number in the last line of the file is the average.

Load Memory: The load memory method allows the user to store the patterns depicted in pointer presentation on a normal sized associative memory.

Verify Memory: As in the previous application, the verify memory option presents the units of the associative memory as a list.

Run Image Set (Normal): Similar to the run image set for the high dimensional memory, this option allows to recall all the patterns presented in a given text file.

Use Image High Dimensional: Through this option the user can recall a single pattern from the high dimensional associative memory.

Create Image Normal: Creates a weight matrix diagram of the normal memory.

Graph Info: Creates the weight matrix distribution of the high dimensional memory.

Create Image High Dimensional: Similar to the option above, creates a weight matrix diagram of the high dimensional memory.

Example

A typical first step in the test sequence would be to load the pattern file created by the random generated application into the normal and high dimensional spase associative memories. The second step is to execute the run image set methods in order to get the average error rate in both memories. The last phase consists on displaying the information in a graph.

Appendix B

Experiment Results Data

Hierarchical Associative Memory

Nº Images	1000	5000	10000	20000	40000	80000
Non Optimized/ms	32	140	180	437	656	1234
Optimized/ms	16	80	120	255	360	720
Non Optimized/operations	112429	558230	1119836	2229912	4451945	8915002
Optimized/operations	97700	484879	972747	1936425	3866646	7742327

B. 1 Table with data for aggregation 2 and unit size 1000 (no correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000	80000
Non Optimized/ms	31	130	188	438	656	1235
Optimized/ms	16	62	80	220	404	710
Non Optimized/operations	112429	558230	1119836	2229912	4451945	8915002
Optimized/operations	95802	474711	953251	1897674	3787924	7586830

B. 2 Table with data for aggregation 5 and unit size 1000 (no correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000	80000
Non Optimized/ms	31	140	219	359	657	1266
Optimized/ms	16	70	130	210	380	715
Non Optimized/operations	112429	558230	1119836	2229912	4451945	8915002
Optimized/operations	92369	458941	920030	1834699	3661161	7331978

B. 3 Table with data for aggregation 10 and unit size 1000 (no correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000	80000
Non Optimized/ms	32	156	225	379	668	1304
Optimized/ms	31	62	109	188	313	469
Non Optimized/operations	112429	558230	1119836	2229912	4451945	8915002
Optimized/operations	83160	414985	832078	1655919	3307075	6621206

B. 4 Table with data for three step aggregation (25, 10,5) and unit size 1000 (no correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	156	328	437	485	485
Optimized/ms	47	156	454	672	531
Non Optimized/operations	1162976	4212167	5882489	6734152	6838586
Optimized/operations	361283	1351090	7754809	8857154	8971920

B. 5 Table with data for aggregation 2 and unit size 1000 (with correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	139	350	420	502	520
Optimized/ms	31	391	500	546	610
Non Optimized/operations	1162976	4212167	5882489	6734152	6838586
Optimized/operations	321911	5129854	6972825	7877557	7986609

B. 6 Table with data for aggregation 5 and unit size 1000 (with correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	131	330	401	460	519
Optimized/ms	31	375	625	547	563
Non Optimized/operations	1162976	4212167	5882489	6734152	6838586
Optimized/operations	271646	4781701	6489447	7346202	7452462

B. 7 Table with data for aggregation 10 and unit size 1000 (with correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	157	340	412	470	501
Optimized/ms	94	234	547	594	594
Non Optimized/operations	1162976	4212167	5882489	6734152	6838586
Optimized/operations	581024	2626954	8834860	9777035	9897944

B. 8 Table with data for three step aggregation (25, 10,5) and unit size 1000 (with correlation / optimal *M* value).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	16	62	156	160	235
Optimized/ms	16	31	170	172	281
Non Optimized/operations	141036	664910	1172844	1825717	2342931
Optimized/operations	67777	264500	1646932	2549702	3240143

B. 9 Table with data for aggregation 2 and unit size 1000 (with correlation / *M* value below optimal).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	16	67	154	167	209
Optimized/ms	16	94	140	250	297
Non Optimized/operations	141036	664910	1172844	1825717	2342931
Optimized/operations	64884	887168	1519516	2272524	2830689

B. 10 Table with data for aggregation 5 and unit size 1000 (with correlation / *M* value below optimal).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	21	75	145	189	200
Optimized/ms	15	94	203	219	266
Non Optimized/operations	141036	664910	1172844	1825717	2342931
Optimized/operations	60803	836877	1402865	2082504	2604960

B. 11 Table with data for aggregation 10 and unit size 1000 (with correlation / *M* value below optimal).

Nº Images	1000	5000	10000	20000	40000
Non Optimized/ms	32	60	124	164	170
Optimized/ms	31	94	218	282	328
Non Optimized/operations	141036	664910	1172844	1825717	2342931
Optimized/operations	142078	730508	2194436	3048741	3640662

B. 12 Table with data for three step aggregation (25, 10,5) and unit size 1000 (with correlation / *M* value below optimal).

Nº Images	100	500	1000	2000
Non Optimized/ms	16	18	47	62
Optimized/ms	8	10	15	31
Non Optimized/operations	24834	106684	193778	394263
Optimized/operations	21414	60798	91725	158775

B. 13 Table with data for aggregation 2 and unit size 1000 (with correlation / optimal *M* value).

Nº Images	100	500	1000	2000
Non Optimized/ms	14	21	46	62
Optimized/ms	10	14	23	32
Non Optimized/operations	24834	106684	193778	394263
Optimized/operations	21054	59183	87774	146288

B. 14 Table with data for aggregation 5 and unit size 1000 (with correlation / optimal *M* value).

Non Optimized/ms	100	500	1000	2000
Optimized/ms	12	21	63	84
Non Optimized/operations	16	15	31	100
Optimized/operations	24834	106684	193778	394263
Optimizado/O	20356	56496	82231	523520

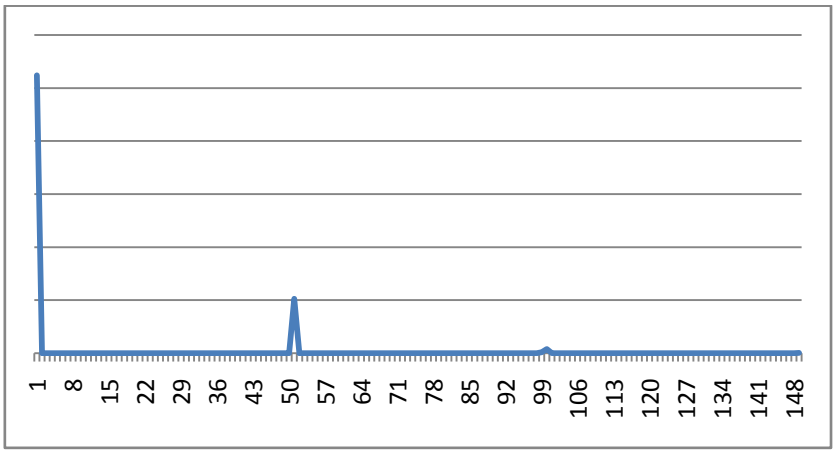
B. 15 Table with data for aggregation 10 and unit size 1000 (with correlation / optimal *M* value).

Nº Images	100	500	1000	2000
Non Optimized/ms	14	19	40	70
Optimized/ms	5	31	32	45
Non Optimized/operations	24834	106684	193778	394263
Optimized/operations	18716	135369	192330	285970

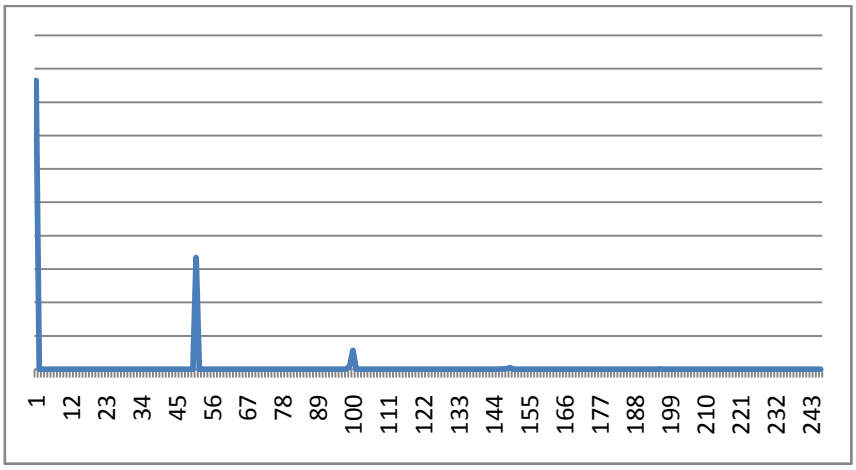
B. 16 Table with data for three step aggregation (25, 10,5) and unit size 1000 (with correlation / optimal *M* value).

Appendix C

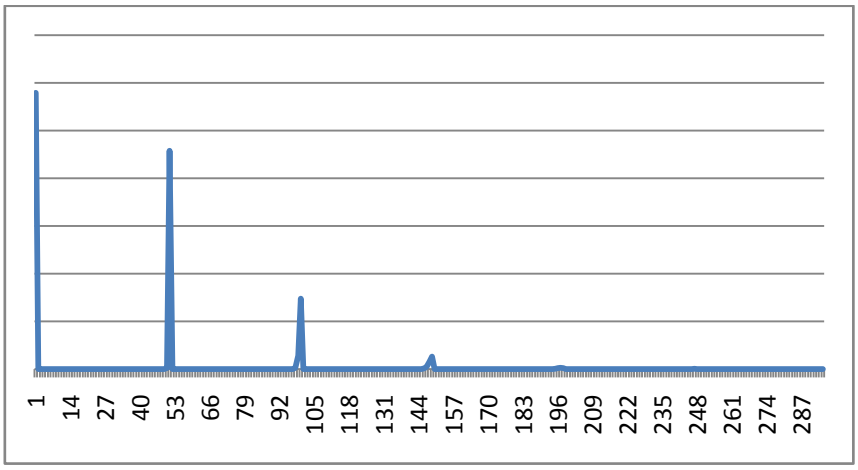
High Dimensional Space Distribution Diagrams



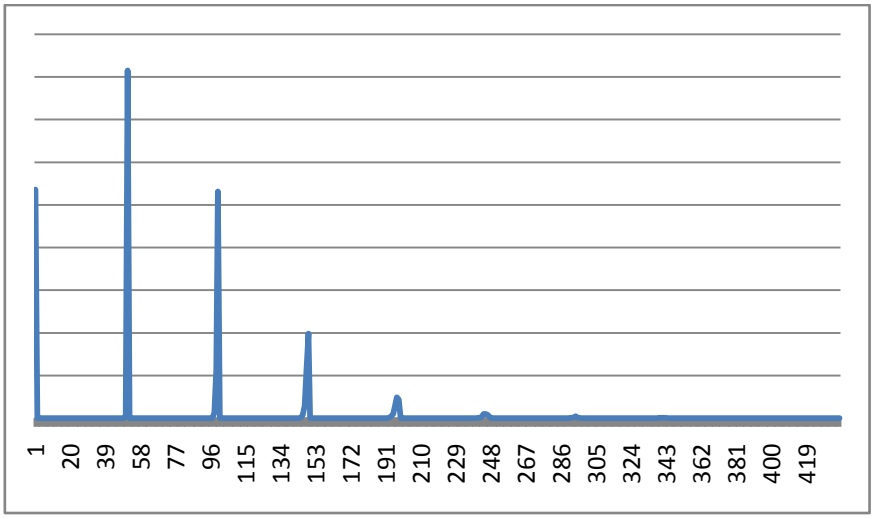
C. 1 Distribution of ones in the high dimensional memory after storing 50 images with fifty percent ones.



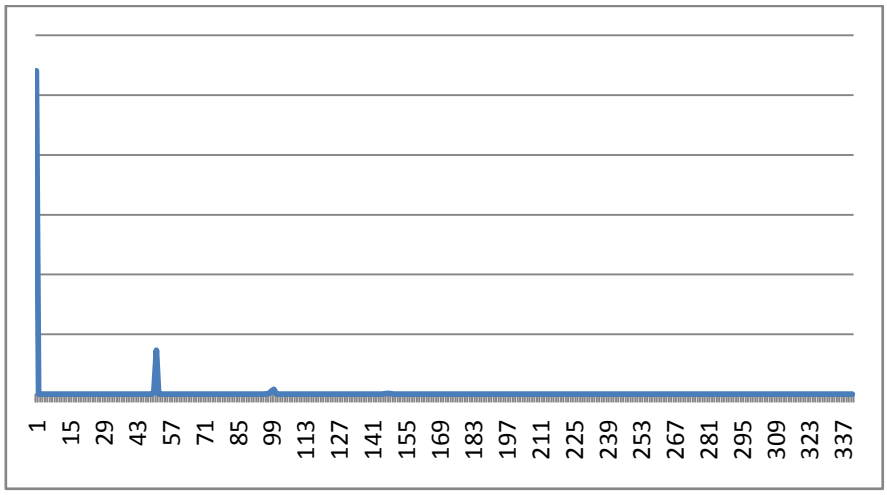
C. 2 Distribution of ones in the high dimensional memory after storing 100 images with fifty percent ones.



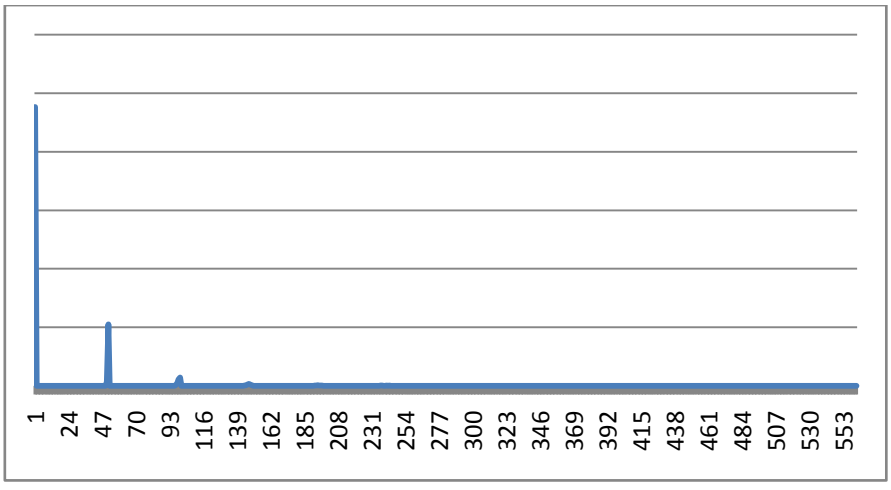
C. 3 Distribution of ones in the high dimensional memory after storing 200 images with fifty percent ones.



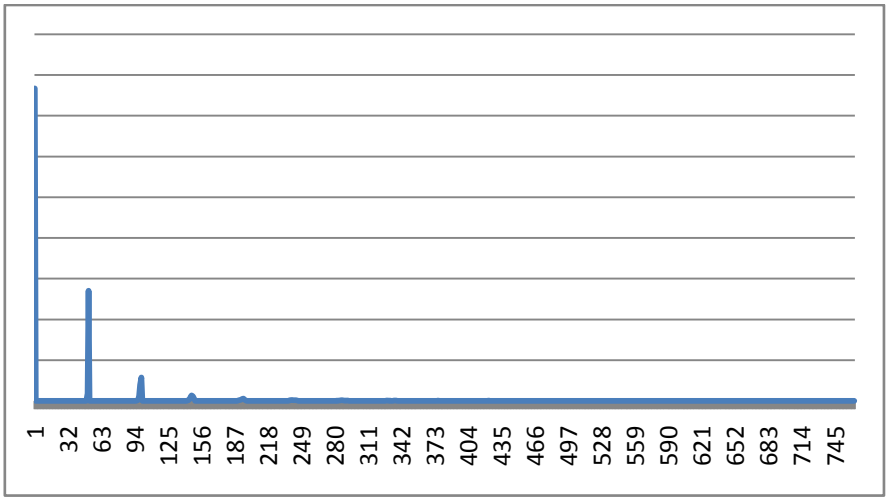
C. 4 Distribution of ones in the high dimensional memory after storing 400 images with fifty percent ones.



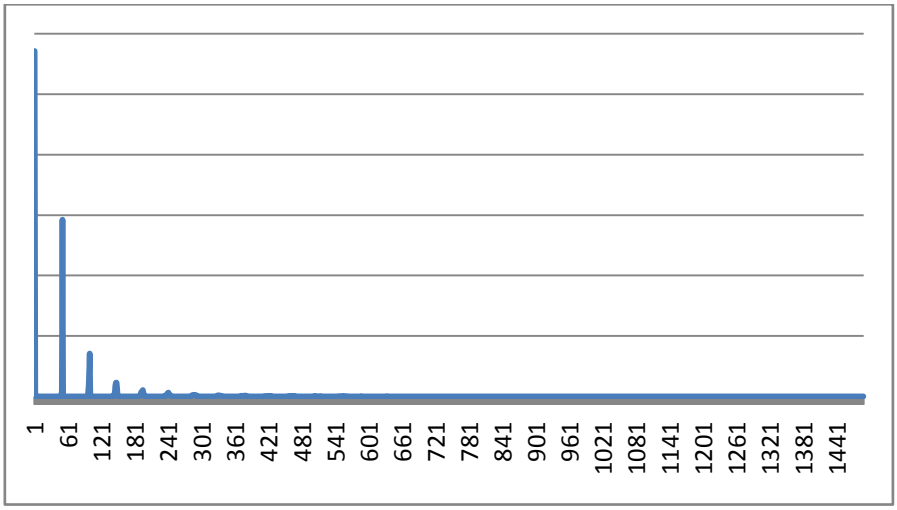
C. 5 Distribution of ones in the high dimensional memory after storing 50 images with thirty percent ones.



C. 6 Distribution of ones in the high dimensional memory after storing 100 images with thirty percent ones.



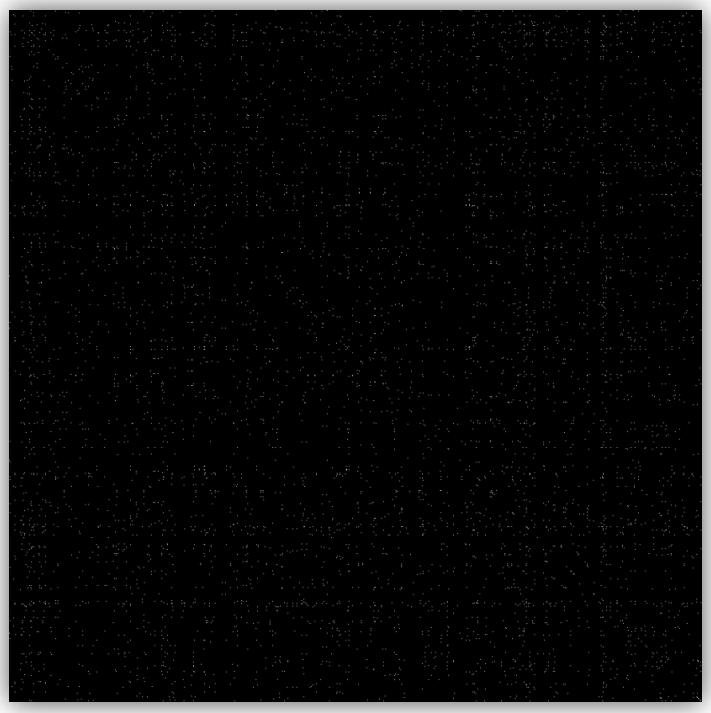
C. 7 Distribution of ones in the high dimensional memory after storing 200 images with thirty percent ones.



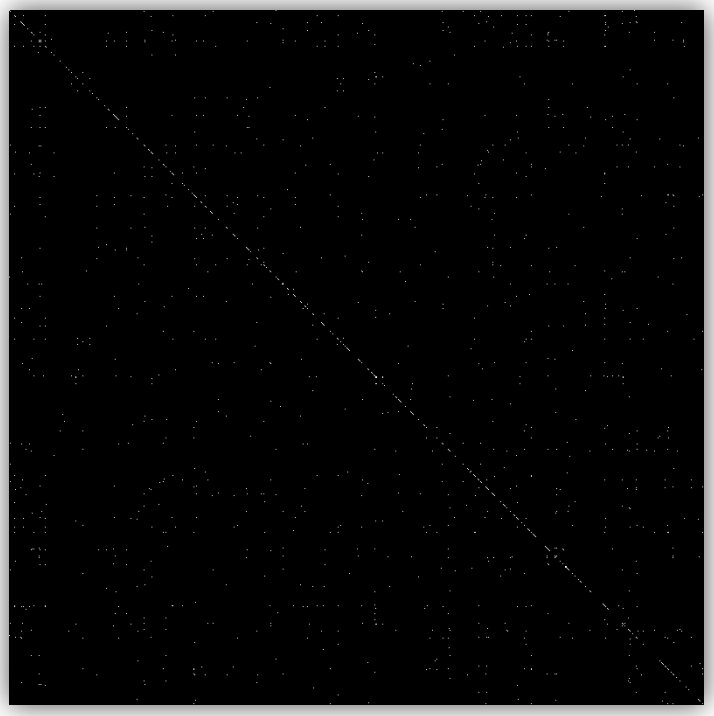
C. 8 Distribution of ones in the high dimensional memory after storing 400 images with thirty percent ones.

Appendix D

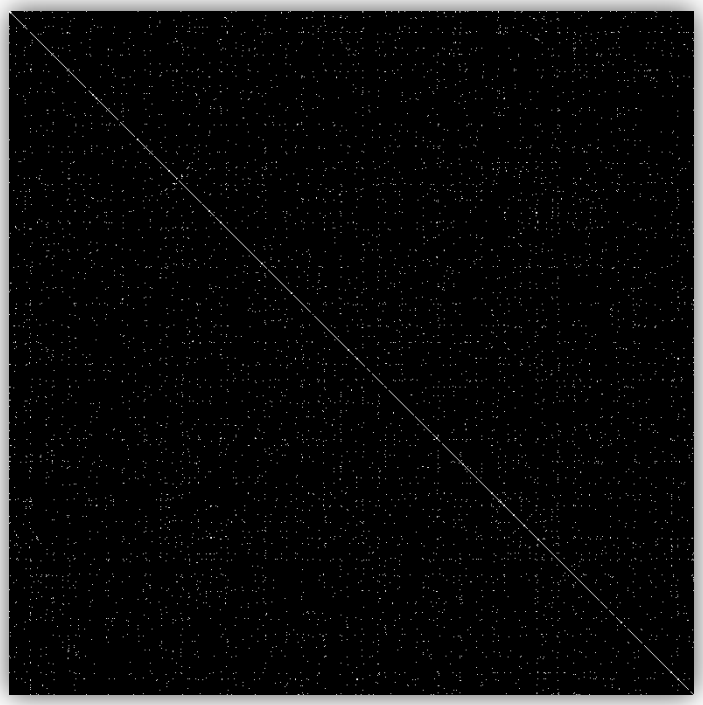
Weight Matrix Diagrams



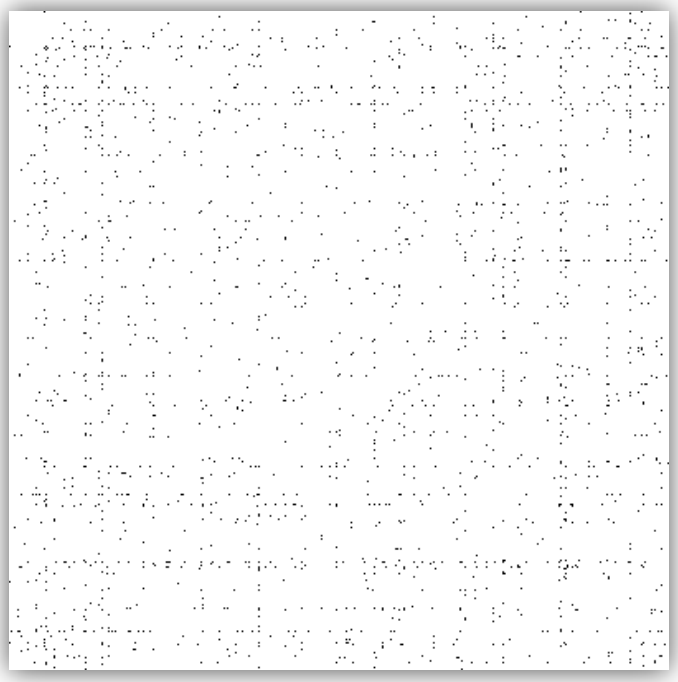
D. 1 Snapshot of the memory after the storage of 1000 images with no correlation.



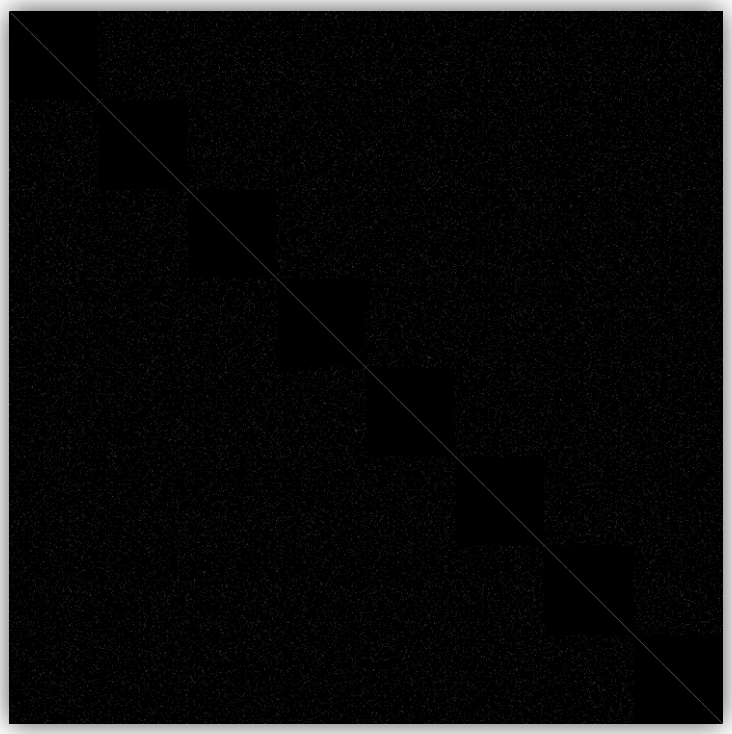
D. 2 Snapshot of the memory after the storage of 100 images with correlation.



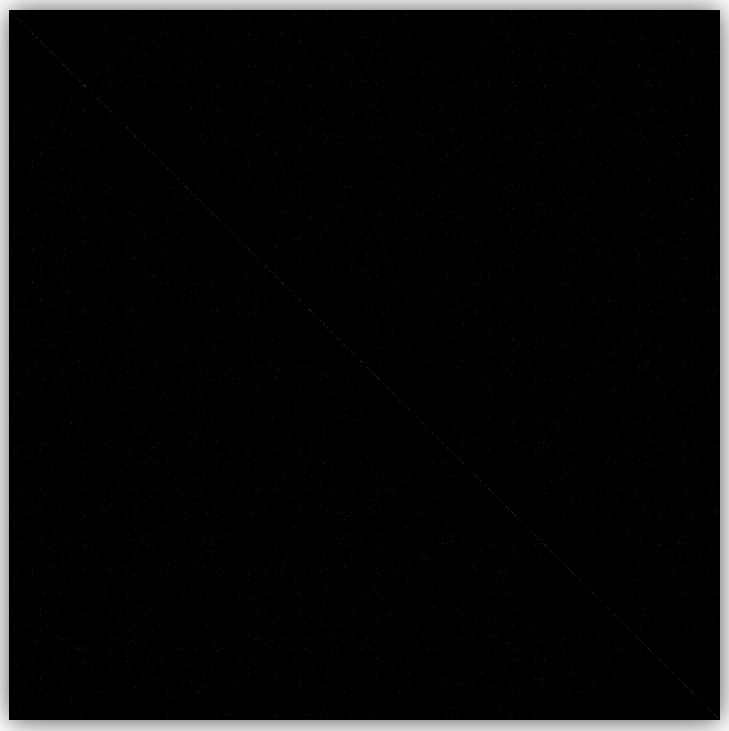
D. 3 Snapshot of the memory after the storage of 500 images with correlation.



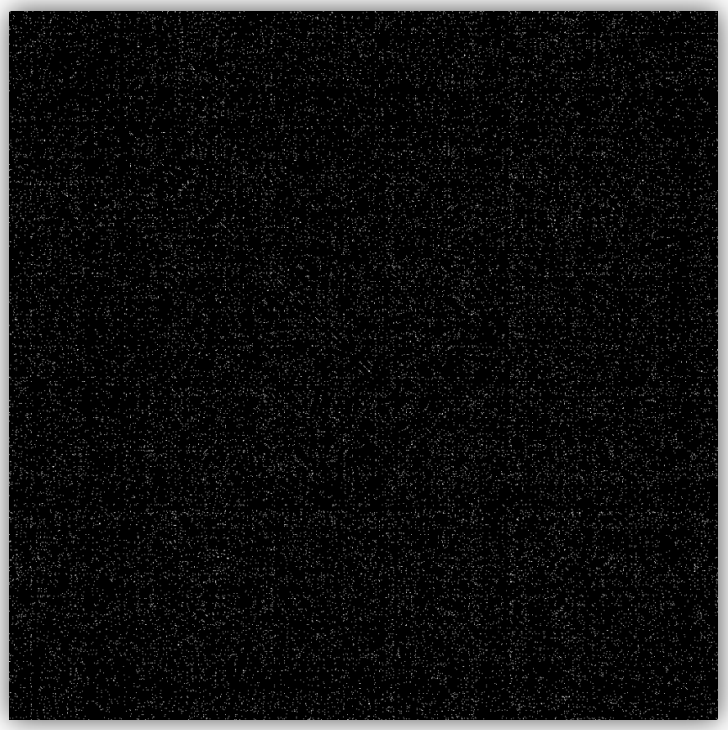
D. 4 Snapshot of the memory after the storage of 50 images with 30 percent ones.



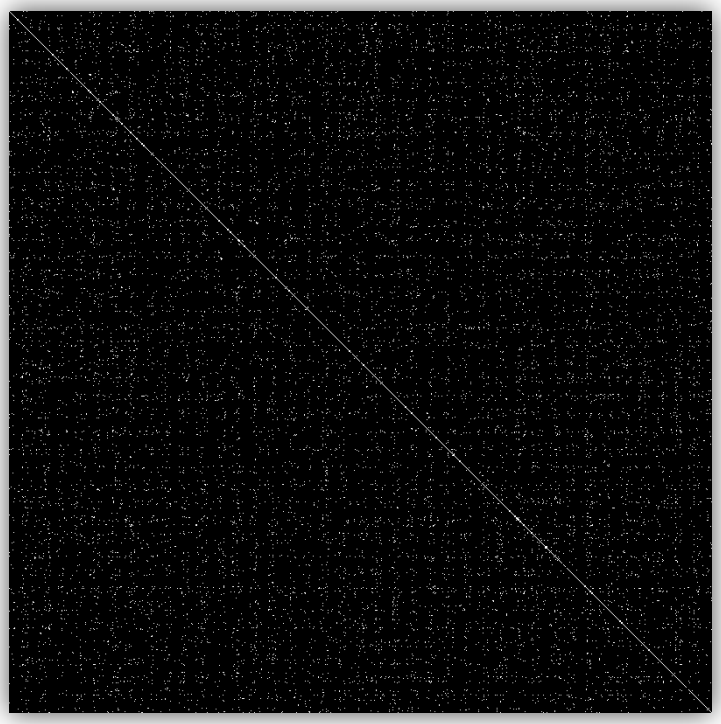
D. 5 Snapshot of the high dimensional memory after the storage of 800 images with 50 percent ones.



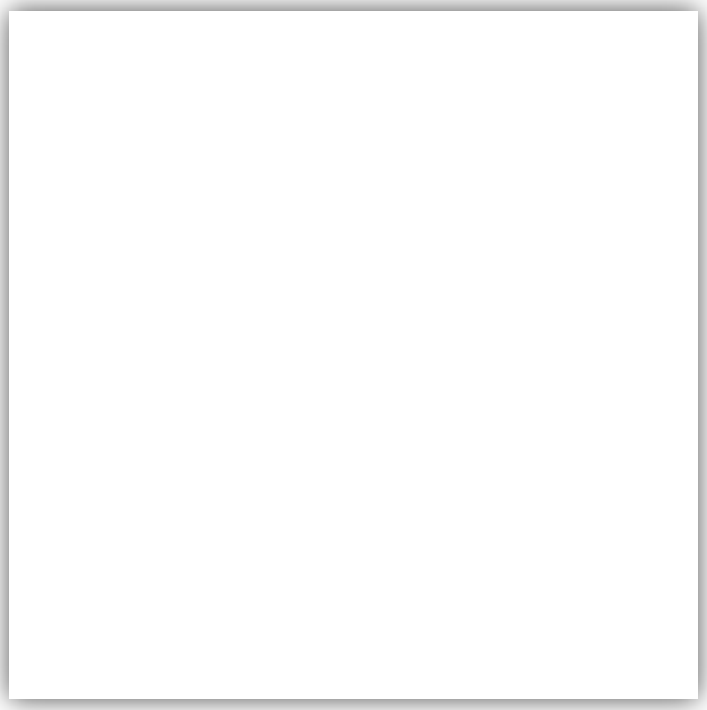
D. 6 Snapshot of the high dimensional memory after the storage of 64 images with 50 percent ones.



D. 7 Snapshot of the memory after the storage of 5000 images with no correlation.



D. 8 Snapshot of the memory after the storage of 1000 images with correlation.



D. 9 Snapshot of the memory after the storage of 50, 100, 200 and 400 images with 50 percent ones.