Efficient Learning using Kernelized Associative Memories

Aitor Ortiz de Latierro Olivella
Bachelor Degree in Informatics Engineering - Computing

supervised by
Lluís A. Belanche Muñoz
Computer Science

January 17, 2017

Contents

List of Tables

List of Figures

1 Introduction

The **associative memory**, defined as the ability to learn and remember relationships between unrelated items, is one of the many types of memories that exist in the human brain. Since 1982 –with the description of what would be known as the Hopfield network– we have tried to replicate this fascinating idea computationally following (or not) the original biological model.

Hitherto, many applications and possible uses have been explored and many ideas have been tried to improve these memory models [14] [9] [1]. Recovering data from incomplete or corrupted versions, associating handwritten letters to their binary representation or even helping to understand how the human brain works are examples of the former. Finding more general structures, developing optimal methods and applying algorithmic tricks to increase their efficiency are examples of the latter.

The goal of this degree final project is to continue this long path a little step further focusing on a recently reopened topic of *kernelized* associative memories. These memories use the mathematical kernel trick that allows them to work implicitly at high-dimensional spaces very efficiently. We develop a learning framework from the scratch using the JULIA language, implement some of the most fundamental associative memories as well as new kernelized variations and compare them in a series of experiments. We focus also on the method's efficiency, something not always pointed out, analyzing critically their cost in memory space and computation time.

From the tests results we have obtained positive feedback about the performance and efficiency of our novel methods as well as instructive insights about the associative memory nature. We explain all of them with theoretical arguments emphasizing the most unintuitive ones.

This thesis is divided in 14 chapters, including a preparatory section of terminology, an extensive overview of past and novel theory, experiments with critical results analysis, and ideas for future projects. It also contains management-oriented chapters, like budget estimations and a sustainability study, meeting the school requirements and wrapping up the project.

2 Background

An **associative memory** (AM) is, by definition, a kind of neural network that learns to relate different patterns without a direct connection. Given a set of p pairs $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$ with $\mathbf{x}_i \in \mathcal{X}, \mathbf{y}_i \in \mathcal{Y}$, the goal of an associative memory (AM) is to store the pairs in such a way that, when presented with a vector, known as a *pattern*, $\mathbf{x}_i$, its corresponding pattern $\mathbf{y}_i$ is correctly recalled, even if the input pattern $\mathbf{x}_i$ has been corrupted in some way. The patterns are normally expressed in the form of bipolar vectors representing the ON and OFF states continuing the neural analogy. We have then:

$$\textit{for every memory } i: \ \mathbf{x}_i \leftrightarrow \mathbf{y}_i \ \textit{with } \mathbf{x} \in \{-1,+1\}^m \ \textit{and } \mathbf{y} \in \{-1,+1\}^n$$

where $\leftrightarrow$ has to be read as “is associated with” For convenience we also define $\mathbf{X} \in \{-1,+1\}^{p\times m}$ and $\mathbf{Y} \in \{-1,+1\}^{p\times n}$ as the matrices that contain all the memories of each type, by rows.

Although the notation makes it a bit unintuitive, it is important to ignore the asymmetric assumption than $\mathbf{x}$ will correspond to the input patterns and $\mathbf{y}$ to the output ones. Rather, the association is bidirectional, and in principle there is no privileged component.

An important classification of AMs is according to the kind of associativity. The scenario above, where the associative memory stores pairs of patterns, is known as a *heteroassociative memory.* The special case where the associative memory stores only a set of single patterns to remember, rather than a set of pairs, is known as the *autoassociative* case, and some associative memories are specially designed for this case, like the well-known Hopfield network [5]. The autoassociative case is equivalent to having $\mathbf{x}_i = \mathbf{y}_i$ for all $i \in \{1, ..., p\}$ in the formulation above.

2.1 Learning Process

It is important to explain the two phases of the learning process that will follow the AMs included in the text, and the majority of machine learning models, as it will require to talk about some relevant concepts. To illustrate the process we will use a simple **linear regression** as an example.

The first one is the **training** phase. In this step we are presented with some training examples and we have to find the values for our model parameters that best adapt to them. What do we mean by **best** and **how** we find them will define different training processes with different properties.

Example: We have some examples, inputs $\mathbf{X}$ and outputs $\mathbf{y}$, and we have to find the best values for the parameters $\mathbf{w}$ of our model $y = \mathbf{w}^{\mathrm{T}}\mathbf{x}$. One possible kind of training for this model is the ordinary least squares estimation that finds the parameters that reduce most the **mean squared error of the training examples**. Fortunately it is closed-form, $\widehat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$.

The second one is the **inference** phase. In this step we *produce* the outputs for different inputs. We can then evaluate our model performance if we know how the output should be or just use it for the application that it was created.

Example: When we are presented with a new input example $\mathbf{x}_{new}$ we just have to compute our model $y_{new} = \hat{\mathbf{w}}^{\mathrm{T}}\mathbf{x}_{new}$ to obtain the predicted output.

In some of the AMs that will be explained in the text the inference phase will not be completely straightforward as they have **recurrent** structure, the output will not be obtained by a number of simple mathematical operations. For these models there will be an iterative inference process with **forward** and **backward** steps and a condition to stop the algorithm.

This also brings the idea of **convergence** of the AMs to the table: we would like that the inference process eventually ends because the stopping criteria was met. In this thesis we will mention, when it has been proven, if those AMs always convernge for different reasonable stopping conditions.

We would also like to note that in some cases these two phases of the learning process are not completely separable, e.g. in **online learning**, but this kind of models are beyond the scope of our work and will not be explained.

2.2 Energy-Based Learning

All the models of AMs that will be presented in this thesis can be seen as a part of the **Energy-Based Models** (EBMs) family. These models basically assign a scalar value, an *energy*, to each possible configuration of input-output variables with lower values for more plausible combinations [8].

In order to memorize the set $\mathcal{S}$ of associated pairs, we first assume there is some general dependency between the components x_i of $\mathbf{x}$ and the components y_j of $\mathbf{y}$. We then want to discover the form of this dependency so that we can use it to map an input vector $\mathbf{x}_i$ from $\mathcal{S}$ to its corresponding vector $\mathbf{y}_i$ (and potentially vice versa). One way to do this is to define first the believed structure, or *architecture*, of the dependencies, and then to define an associated function, known as the *energy function*, that takes $\mathbf{x}$ and $\mathbf{y}$ as inputs and outputs a measure of how well $\mathbf{x}$ and $\mathbf{y}$ satisfy the dependency.

The process of optimizing the parameters of the architecture (or energy function) such that the energy function indicates high compatibility between vector associations in the list is known as *training.* Finally, with the optimized model in hand, we need a way to *infer* $\mathbf{y}$ given only $\mathbf{x}$. This process of defining an architecture, an associated energy function, a training method/loss function, and an inference method is known as *energy-based modeling.*

With this fact in mind we can reduce the model to an energy function, the training step to the search of parameters that minimize that energy function for our training examples, and the inference step to the search of the minimum-energy combination of outputs.

2.3 Kernel Methods

As we will see in during the thesis (viz. section 6.2), one of the novel ideas is the **kernelization** of a particular AM learning process. In order to understand that, we will need to explain what we actually mean by kernelization. As we explained in 2.1 most of the machine learning algorithms can be seen as the mapping of some data $\mathbf{x}$, a sample of some features, to an output $\mathbf{y}$. The effectiveness of our method will depend both on the **expressiveness** of the algorithm, how much it can transform our data, and on the **quality** of our data, how much it is related with the desired outputs.

On the one hand, improving the first factor using more general and powerful algorithms can become computationally intractable and we can become prone to **overfitting**. On the other hand increasing the quality of the data is in general out of the question in most problems.

Kernel Methods are basically augmentations of simple algorithms –via an implicit mapping function– provided they can be expressed with only inner products between data instances. These algorithms, that are originally *linear*, can be non-linearized by what it is called **kernel trick**, changing that vector inner product for a kernel function. To explain this, it is best to consider a specific example that will be developed below.

Consider the problem of fitting a linear function $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ to a dataset $\mathcal{S} = \{(\mathbf{x}_i, y_i)\}$ for $i \in \{1, ..., p\}$, with $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \mathbb{R}$, such that $f(\mathbf{x}_i) \approx y_i$ in the mean square sense, and such that f makes good predictions on new inputs (i.e., it should *generalize* well).[1] Classical *ridge regression* does this by finding the $\mathbf{w}$ that minimizes the sum of the squared errors over the dataset, plus a regularization term that penalizes complexity by pushing the Euclidean length of the weight vector $\mathbf{w}$ to 0:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} E_\lambda(\mathbf{w})$$

where

$$E_\lambda(\mathbf{w}) = \sum_{i=1}^{p}(y_i - \mathbf{w}^{\mathrm{T}}\mathbf{x}_i)^2 + \lambda \left\|\mathbf{w}\right\|^2$$

The optimal solution for this turns out to be

$$\hat{\mathbf{w}} = (\mathbf{X}^{\mathrm{T}}\mathbf{X} + \lambda\mathbf{I}_n)^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}$$

where $\mathbf{X} \in \mathbb{R}^{p\times n}$ is the matrix with row vectors $\mathbf{x}_i$, $\mathbf{y}$ is the vector of the y_i and $\mathbf{I}_n$ is the $n \times n$ identity matrix. After some algebra, it turns out that we can re-write $\hat{\mathbf{w}}$ as a linear combination of the input vectors:

$$\hat{\mathbf{w}} = \sum_{i=1}^{p}\alpha_i\mathbf{x}_i, \text{ with } \boldsymbol{\alpha} = (\mathbf{X}\mathbf{X}^{\mathrm{T}} + \lambda\mathbf{I}_p)^{-1}\mathbf{y}$$

[1] It is costumary to include a bias term w_0; to do this we may just add one dimension to the $\mathbf{x}_i$ as $\mathbf{x}_i := (1, \mathbf{x}_i)^{\mathrm{T}}$ and then $\mathbf{w} := (w_0, \mathbf{w})^{\mathrm{T}} \in \mathbb{R}^{n+1}$.

So $f(\mathbf{x})$ can be written as

$$f(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i {\mathbf{x}_i}^{\mathrm{T}} \mathbf{x}$$

Here's where the trick comes in: the data from the input space *only appears in inner products with other vectors from the input space.* To see why this is useful, assume we now want to make the transformation from the input space into some higher dimensional *feature space*. Let $\phi : \mathcal{X} \to \mathcal{H}$ be such a transformation (known as the *feature map*), where $\mathcal{X}$ is the domain of the input data ($\mathbb{R}^n$ in the case of ridge regression) and $\mathcal{H}$ is some Hilbert space (which has the property that the inner product between vectors in the space is well-defined). We can then substitute $\phi(\mathbf{x})$ for $\mathbf{x}$ in all positions in the solution for $f(\mathbf{x})$ (and in the matrix $\mathbf{X}$ used in the computation of $\boldsymbol{\alpha}$):

$$f(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i \phi(\mathbf{x}_i)^{\mathrm{T}} \phi(\mathbf{x})$$

As mentioned before, we do not want to have to define the transformation ϕ explicity. For a certain class of functions between two vectors –known as *kernel functions*– the evaluation of the function is guaranteed to be equivalent to performing an inner product between transformations of those vectors in some Hilbert space. Without defining ϕ explicitly, we can replace the inner products with a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for which the identity $k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{y})$ holds, so that

$$f(\mathbf{x}) = \sum_{i=1}^{p} \alpha_i k(\mathbf{x}_i, \mathbf{x})$$

and $\boldsymbol{\alpha}$ becomes

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_p)^{-1} \mathbf{y}$$

where $\mathbf{K}$ is the $p \times p$ *kernel matrix* with elements $(\mathbf{K})_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. Performing the replacement of the inner product with a kernel function makes it possible to work directly with the input data while still effectively working with a transformation of the data, and is known as the *kernel trick*.

A couple of very common kernels are

- **Polynomial kernel**: $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{\mathrm{T}} \mathbf{y} + b)^d$, with $d \in \mathbb{N}^+, b \geq 0, d \in \mathbb{N}$
- **Gaussian RBF kernel**: $k(\mathbf{x}, \mathbf{y}) = \exp(-\gamma \left\| \mathbf{x} - \mathbf{y} \right\|^2), \gamma > 0$

3 Problem Formulation

After a brief introduction of the topic it is important to define the project that this thesis is about. We need to explain its scope, the goals and limitations, and its associated risks in order to assess its success at the end of it.

3.1 Objectives

Although the goal of this thesis is pushing forward the state of the art of kernelized AM's, focusing mostly on efficient methods, we can divide its contributions in different objectives. Having these concrete goals makes more clear their progress and helps to structure the project.

1. Although it is really only a prerequisite for the core work of this thesis, the development from scratch of a **flexible framework to implement different types of AMs** will be a considerable chunk of the project. We want to make clear that the creation of a new environment, although more demanding, was more reasonable than the reutilization of a different one. The previously existing ones were too generic, not focused on associative memories, or not flexible enough to implement the new developed AM's variants or some of the planned experiments. As a subgoal, all the programming will be done in JULIA [2] a relatively new language for technical computing. We will explain more about the reason of this decision in the next section **Project Methods & Tools**.

2. **Novel AM Kernelization**: one of the ideas that started off the project was the kernelization of an AM learning process that had not been done yet in an heteroassociative context, the *pseudoinverse training*. This derivation requires some extra steps compared to a standard kernelization, and fills a gap in previous literature. It has to be said that this method was not developed for efficiency at all but we thought that it was important to include in the project.

3. **Sparse Learning**: An important objective of the project, and the one that gives its name, is the development of new efficient learning methods for associative memories. In order to achieve this efficiency we will focus on sparsity, aimed at discarding redundant memory associations in order to make the inference process faster.

4. **Experimental part**: An important co-requisite that deserves being counted on as a goal is the development of a series of experiments to compare the different algorithms and methods. Although this project has a motivating theoretical component, we would not like to forget about practice, the numerical results. Some of the experiments will also have a secondary goal, to show the capabilities of kernelized associative memories in tasks different from the purely associative one.

In this thesis we focus on efficiency apart from efficacy, by creating **Sparse Kernelized Associative Memories** that benefit from the training memories redundancy. In particular, we follow the path started by [12] proposing a more general kernelization in **Kernelized Pseudoinverse Memories**. We also give importance to the fact that the memories can be *heteroassociative*, we think that an AM has to be seen symmetrically, from $\mathbf{x}$ to $\mathbf{y}$ and from $\mathbf{y}$ to $\mathbf{x}$. Finally, although not patent in the theoretical core of the thesis, the development of our own framework is an important part in itself. The use of a novel language like JULIA and the re-implementation of all the explained memories to be flexible enough for experimentation justifies it.

3.2 Limitations

Every project has some fundamental bounds in the available resources that end up defining an scope. We have to take into account that this project is a bachelor thesis worth 18 ECTS credits (roughly 450 work hours).

Limited Experiments: One of the biggest limitations is the **computational power**. Many interesting experiments require too much time to be done with some statistical validity because the AMs, and in particular the kernelized ones, are not fast.

Baseline Diversity: The comparison of new methods with preset baselines is a useful way to check their progress. In this project we compare them to different associative memories, algorithms of the same kind, but we don't compare them with different classes of machine learning algorithms.

This limitation is mainly because of schedule reasons, the implementation of different algorithms for my experiments would be expensive in time and would not allow to parts more related with the project.

Alternative Methods: During the brainstorming sessions we had more ideas than the ones finally implemented in the paper. Some were discarded because of their impracticability and others just because we did not choose them.

At the end of the document, section **Future Work**, the most interesting discarded ideas are presented as different paths to be continued.

3.3 Risks

It is natural to evaluate the associated project risks to take them into account when we are planning the project. Their economical evaluation and the preventive actions that have been taken will be explained in later sections.

Now we will name the 3 most important risks.

Method Failure: Although sound in theory, some of the new developed methods may not work in some cases in practice. It is important to document it and analyze the motives of the failure, which will help to understand more about the topic and avoid that future research projects fall in this pitfall.

One could say that this is not a real risk, as it is just a possibility when testing something new, but as it is not the expected outcome we think that it is important to consider its consequences.

Expensive Experimentation: Some of the planned experiments, although already filtered as we say in 3.2, may still be too expensive in terms of **computer power**.

Time Problems: One of the most recurrent problems in projects is temporal cost estimation of the tasks. We have a limited temporal budget and we have to try to achieve all the planned objectives without exceeding it. An initial realistic plan with some flexibility is **essential** for a smooth and successful project.

All this will be fully explained in the **Work Plan** section.

3.4 Previous Work

In this section we will review the existing literature about associative memories with emphasis on the ones that talk about their kernelization. Although these memories can be seen as a type of neural network we will not be that general as it would require an enormous amount of effort to review all the field. We encourage the reader to check the elaborate review [13].

3.4.1 Associative Memories.

The concept of associative memory comes from the 80s with the **Hopfield Network** and the **Bidirectional Associative Network** (BAM) by B. Kosko [7]. Both are recurrent networks, but the former was autoassociative and the latter heteroassociative. From then numerous articles and publications have been written analyzing their capacity, generalization abilities as well as different training and inference methods. Their use in different kinds of data and their behavior against different types of noises has also been extensively studied. The most relevant papers for this thesis have been:

B. Kosko [7] From the BAM's creator, a crucial article about heteroassociative memories and bidirectionality, recurrence.

Y.-J. Jeng et al. [6] An interesting paper that starts to explore non-linearities in order to improve the AM precision (as we do in kernelization).

We recommend [1] for a complete review about the topic.

3.4.2 Kernelized Associative Memories.

The application of some kind of kernel trick to associative memories it is not new although it can be said to be a current research topic. We could highlight because of their relevance with our work:

B. Caputo et al. [3] Modelling the problem as Markov network this is the first paper that kernelized the energy function. Despite this, it is only focused in the autoassociative case and the use of that energy function as a patter classifier, they do not study the inference process.

D. Nowicki et al. [10] A very thorough thesis that reviews autoassociative and heteroassociative memories and proposes a kernelization for an autoassociative version. They focus on the creation of flexible memories for online learning, something out of the project scope.

M. Saltz [12] Master thesis that reviews that studies the BAM kernelization in detail. He also proposes new inference methods based on classical mathematical optimization techniques like Hill Climbing.

4 Project Methods & Tools

In this section we will explain some key insights about the followed project methodology as well as the tools used in its development. It is important to document this processes as lessons for future ventures can be extracted.

4.1 Development Cycle

It is important to remark that this project is a solo project in its development with only a tutoring figure to guide and advise. Many of the team methodologies learned through the degree do not apply as real parallel work is hardly ever possible because of this limitation.

The project was roughly divided in small tasks and goals and their progress was tracked in weekly meetings with the tutor. Sometimes, if for example not enough progress was achieved or there was some kind of impediment, a meeting could be canceled in exchange of feedback through email.

In case there was need to change the planning the developer proposed the change to the tutor and he approved or recommended distinct options.

4.2 Results Validation

The validation of results is a fundamental part of the project as we think that transparency and reproducibility in the research community is essential.

First, all the experiments will be repeated several times (and not cherry-picked) in order to obtain statistical sound results. The number of repetitions will be specified in each particular case depending on how much time-demanding they are or if they are more or less volatile.

As an external verification, out of the scope of this project, I will also update the code with a polished version of the developed framework with easy commands to try the experiments and check the obtained results.

4.3 Development Tools

This project will use **GitHub** as the basic management and version control system of the code. For the less technical part of the management we will use the **gantter** web app, a free and simple alternative to Microsoft Project.

As we mentioned previously, all the code will be in JULIA, a relatively new but very powerful open-source language for technical computing. We will use only some basic external packages like PYPLOT for plotting.

This document and all the external documentation has been written using **LaTeX** with the cross-platform editor **TexMaker** and several packages.

5 Associative Memories

Having briefly defined in the **Background** section what an AM is we now can explain in detail some of their main implementations.

First of all, taking into account the EBMs framework used in this thesis, we need to define the energy function. The most natural option is:

$$E(\mathbf{x}, \mathbf{y}) = -\frac{1}{2}(\mathbf{x}^{\mathrm{T}}\mathbf{W}_1\mathbf{y} + \mathbf{y}^{\mathrm{T}}\mathbf{W}_2\mathbf{x})$$

where lower values are associated with more compatible pairs of vectors. Moreover, in order to understand why we separate the function in matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ it is important to see in Figure 1 how an AM looks like..

5.1 Linear Associative Memory

The simplest AM is the so-called **Linear Associative Memory** (LAM).

By definition it only has forward connections ($\mathbf{W}_1$), the backward connections $\mathbf{W}_2$ are set to zero. We end up with this simple energy function:

$$E(\mathbf{x}, \mathbf{y}) = -\mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{y}$$

Because of its nice structure, we can infer the most compatible $\mathbf{y}$ easily and obtain the inference rule:

$$\mathbf{y}^* = \underset{\mathbf{y}\in\{-1,+1\}^n}{\arg\min}\ E(\mathbf{x}, \mathbf{y}) = \operatorname{sgn}\left(-\frac{\partial E}{\partial \mathbf{y}}\right) = \operatorname{sgn}(\mathbf{x}^{\mathrm{T}}\mathbf{W})$$

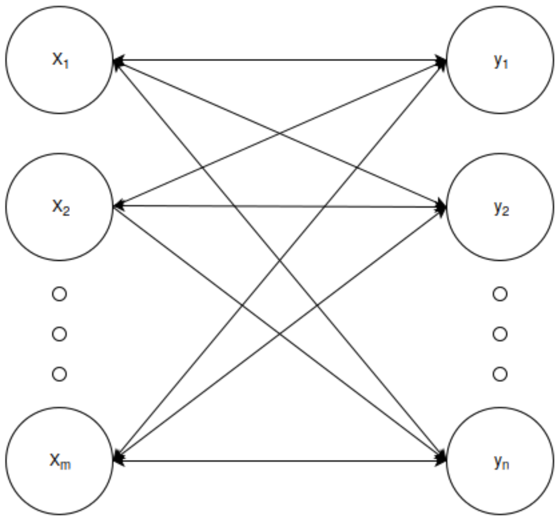


Figure 1: Associative Memory structure. The arrows to the right represent $\mathbf{W}_1$ connections and the arrows to the left represent $\mathbf{W}_2$ connections.

5.1.1 Hebbian Training

The first developed training method for LAMs is named after Hebbian theory in neuroscience [4] and it is based on **correlations**.

$$\mathbf{W} = \mathbf{X}\mathbf{Y}^{\mathrm{T}} = \sum_{i=1}^{p} \mathbf{x}_i {\mathbf{y}_i}^{\mathrm{T}} \tag{1}$$

Which basically adds up the training data ($\mathbf{X}$ and $\mathbf{Y}$) correlations in order to set a low energy when we are presented with those memories together again.

$$E(\mathbf{x}_i, \mathbf{y}_i) = -{\mathbf{x}_i}^{\mathrm{T}}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\mathbf{y}_i = -(\sum_{j=1}^{p} {\mathbf{x}_i}^{\mathrm{T}}\mathbf{x}_j)(\sum_{j=1}^{p} {\mathbf{y}_j}^{\mathrm{T}}\mathbf{y}_i)$$

Note that the energy will at least have big negative value for the case $i = j$.

This method has some properties that makes it useful in some situations.

- It has a very **low complexity** as only a matrix multiplication is needed, naively $\mathcal{O}(mnp)$, to obtain the model parameters.
- It allows easy **online learning**, updating the model as data arrives is as simple as adding the correlation matrix of the new association to eq. (1).

5.1.2 Pseudoinverse Training

In the previous training method we did not try to find optimal values and were only guided by correlations between the given training memories. This hints us that maybe there exists a more accurate method to find the parameters $\mathbf{W}$, taking into account all the memories as a whole.

We define the **Optimal Linear Associative Memory** (OLAM) as the LAM whose parameters are found with the expression:

$$\mathbf{W} = \mathbf{X}^{+}\mathbf{Y}^{\mathrm{T}}$$

where $\mathbf{X}^{+}$ stands for the pseudoinverse of $\mathbf{X}$. We say that it is **optimal** as this expression comes from the well known problem of linear least squares regression, summarized below.

We want $\mathbf{XW} \approx \mathbf{Y}$ so for $\mathbf{W}^{*} = \arg\min_{\mathbf{W}} \|\mathbf{XW} - \mathbf{Y}\|^2 = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{Y}$.

OLAM's advantage over Hebbian is its implicit orthogonalization of the input patterns that the pseudoinverse applies. It is shown in detail at [11] but it is instructive to check that in the extreme situation where all the input patterns are orthogonal both methods generate identical weights:

$$\mathbf{W} = \mathbf{X}^{+}\mathbf{Y} = \mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1}\mathbf{Y} = \mathbf{X}^{\mathrm{T}}\mathbf{I}^{-1}\mathbf{Y} = \mathbf{X}^{\mathrm{T}}\mathbf{Y}$$

since ${\mathbf{x}_i}^{\mathrm{T}}\mathbf{x}_j = 1$ if $i = j$ and ${\mathbf{x}_i}^{\mathrm{T}}\mathbf{x}_j = 0$ otherwise.

We have to be more careful with the number of patterns in this memory as the training complexity is higher because we need to find $\mathbf{X}^{+}$.

5.1.3 Loss-based Learning

The OLAM is only optimal to minimize the mean square energy of the memories to remember. Using different criteria that define a different loss we can obtain $\mathbf{W}$ through gradient descent or a similar optimization technique.

5.2 Bidirectional Associative Memory

A **Bidirectional Associative Memory** or BAM [7] is a LAM adding a constrained backward pass. Specifically, it has to be symmetrical to the forward one or, technically, $\mathbf{W}_2 = {\mathbf{W}_1}^{\mathrm{T}} = \mathbf{W}$.

The addition of this backward pass transforms the inference process from a simple closed-form expression to an iterative process until convergence. From now on we will use the notation $\mathbf{s}_{\mathbf{x}}(t)$ and $\mathbf{s}_{\mathbf{y}}(t)$ when we talk about the input and output state at the step t of that iterative process.

The inference in a BAM will then be:

$$\begin{aligned}
\mathbf{s}_{\mathbf{x}}(0) &= \mathbf{x} \\
\mathbf{s}_{\mathbf{y}}(1) &= \mathrm{sgn}(\mathbf{s}_{\mathbf{x}}(0)\mathbf{W}) \\
\mathbf{s}_{\mathbf{x}}(2) &= \mathrm{sgn}(\mathbf{W}\mathbf{s}_{\mathbf{y}}(1)^{\mathrm{T}}) \\
&\dots \\
\mathbf{s}_{\mathbf{y}}(t+1) &= \mathrm{sgn}(\mathbf{s}_{\mathbf{x}}(t)\mathbf{W}) \\
\mathbf{s}_{\mathbf{x}}(t+2) &= \mathrm{sgn}(\mathbf{W}\mathbf{s}_{\mathbf{y}}(t+1)^{\mathrm{T}})
\end{aligned}$$

We say that the process has converged when we enter into a fixed point, and we use that $\mathbf{y}$ or $\mathbf{x}$ (depending on our objective) as the output.

Because of the forward-backward symmetry we will have to be careful when we choose the training method as not all of them will work.

5.2.1 Hebbian Training

Hebbian training was the original method devised for BAM's, the symmetric structure of correlation matrices fits perfectly the constraint.

$$\mathbf{W}_1 = \mathbf{X}\mathbf{Y}^{\mathrm{T}} \textit{ and } \mathbf{W}_2 = \mathbf{Y}\mathbf{X}^{\mathrm{T}} \textit{ satisfies } \mathbf{W}_2 = {\mathbf{W}_1}^{\mathrm{T}}$$

Fortunately, it has the same training complexity than its LAM counterpart.

5.2.2 Pseudoinverse Training

Pseudoinverse training, on the other hand, can not be successfully adapted into this BAM architecture.

$$\mathbf{W}_1 = \mathbf{X}^{+}\mathbf{Y} \textit{ and } \mathbf{W}_2 = \mathbf{Y}^{+}\mathbf{X} \textit{ does not satisfy } \mathbf{W}_2 = {\mathbf{W}_1}^{\mathrm{T}} \textit{ in general}$$

This fact is not mentioned in some previous literature and can be forgotten when implementing associative memories. We think that it is important to emphasize it and we will show what would happen if we implemented this structure anyway in the **Experiments** section.

5.3 Recurrent Associative Memory

The last generalization possible is to forget about the symmetrical constraint and fully use all model parameters $\mathbf{W}_1$ and $\mathbf{W}_2$.

The inference process will change slightly with respect to the BAM one:

$$\begin{aligned}
\mathbf{s_x}(0) &= \mathbf{x} \\
\mathbf{s_y}(1) &= \mathrm{sgn}(\mathbf{s_x}(0)\mathbf{W}_1) \\
\mathbf{s_x}(2) &= \mathrm{sgn}(\mathbf{W}_2\mathbf{s_y}(1)^{\mathrm{T}}) \\
&\dots \\
\mathbf{s_y}(t+1) &= \mathrm{sgn}(\mathbf{s_x}(t)\mathbf{W}_1) \\
\mathbf{s_x}(t+2) &= \mathrm{sgn}(\mathbf{W}_2\mathbf{s_y}(t+1)^{\mathrm{T}})
\end{aligned}$$

As Hebbian training with this structure becomes only a BAM we will just consider the pseudoinverse case with $\mathbf{W}_1 = \mathbf{X}^{+}\mathbf{Y}$ and $\mathbf{W}_2 = \mathbf{Y}^{+}\mathbf{X}$.

6 Kernelized Associative Memories

We explained in **Kernel Methods** that kernelization, if well applied, can transform a simple machine learning into a much more powerful one. Sometimes the possibility of kernelization is clear, as we showed in our KPCA example, but sometimes a little bit of processing is actually needed.

We have that all the AMs explained in the previous sections can be converted to variants of **Kernelized Associative Memories** (KAM). We will separate them by the learning method in order to distinguish more clearly the one of the additions of this thesis, the one that will be known as **KOLAM**.

6.1 Kernelized Hebbian Memories

In this section we will tackle the kernelization of a BAM (**KBAM**) and its particular case of no backward pass **KLAM**. The first and most important step is the kernelization of the energy function that guides our memory.

$$E(\mathbf{x},\mathbf{y}) = -\mathbf{x}^{\mathrm{T}}\mathbf{W}\mathbf{y} = -\mathbf{x}^{\mathrm{T}}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\mathbf{y} = -\sum_{i=1}^{p}(\mathbf{x}^{\mathrm{T}}\mathbf{x}_i)(\mathbf{y}^{\mathrm{T}}\mathbf{y}_i)$$

Now we have an expression that only depends on a scalar product so:

$$E(\mathbf{x},\mathbf{y}) = -\sum_{i=1}^{p} k_{\mathbf{x}}(\mathbf{x},\mathbf{x}_i)k_{\mathbf{y}}(\mathbf{y},\mathbf{y}_i)$$

It is important to note that $k_{\mathbf{x}}$ and $k_{\mathbf{y}}$ do not have to be the same. If we now kernelize the inference process we arrive at the following expressions expressions for the forward and backward pass [12]:

$$\mathbf{s}_{\mathbf{y}}(t+1) = \operatorname{sgn}\left(\sum_{i=1}^{p} k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}}(t),\mathbf{x}_i)\frac{\partial k_{\mathbf{y}}}{\partial \mathbf{s}_{\mathbf{y}}}(\mathbf{0},\mathbf{y}_i)\right) \tag{2}$$

$$\mathbf{s}_{\mathbf{x}}(t+1) = \operatorname{sgn}\left(\sum_{i=1}^{p} \frac{\partial k_{\mathbf{x}}}{\partial \mathbf{s}_{\mathbf{x}}}(\mathbf{0},\mathbf{x}_i)k_{\mathbf{y}}(\mathbf{s}_{\mathbf{y}}(t),\mathbf{y}_i)\right) \tag{3}$$

With this new structure we have to make an evaluation about the changes:

- We have decreased training complexity to zero, we do not have to create any $\mathbf{W}_1$ now $\mathbf{W}_2$ matrices. We can start recalling instantly.
- We have increased **considerably** the computationally cost of inference, each step will require $\mathcal{O}(pm)$ or $\mathcal{O}(pn)$ operations!
- We now have to store **all** the training memories to do inference.

6.2 Kernelized Pseudoinverse Memories

The Kernelized Pseudoinverse Memory, **KOLAM** the direct version and **KRAM** the recurrent one, will be the most powerful memory explained in this text and, according to our data, it has never been explored before with this generality. First of all, as we have done above, we need to kernelize the energy function of the associative memory.

$$\begin{aligned}
E(\mathbf{s}_\mathbf{x}, \mathbf{s}_\mathbf{y}) &= -\frac{1}{2}(\mathbf{s}_\mathbf{x}{}^{\mathrm{T}}\mathbf{W}_1\mathbf{s}_\mathbf{y} + \mathbf{s}_\mathbf{y}{}^{\mathrm{T}}\mathbf{W}_2\mathbf{s}_\mathbf{x}) \\
&= -\frac{1}{2}(\mathbf{s}_\mathbf{x}{}^{\mathrm{T}}\mathbf{X}^{\mathrm{T}}(\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-1}\mathbf{Y}\mathbf{s}_\mathbf{y} + \mathbf{s}_\mathbf{y}{}^{\mathrm{T}}\mathbf{Y}^{\mathrm{T}}(\mathbf{Y}\mathbf{Y}^{\mathrm{T}})^{-1}\mathbf{X}\mathbf{s}_\mathbf{x}) \\
&= -\frac{1}{2}(\sum_{i=1}^{p}\sum_{j=1}^{p} k_\mathbf{x}(\mathbf{s}_\mathbf{x}, \mathbf{x}_i)\mathbf{K}^{-1}_{x,ij}k_\mathbf{y}(\mathbf{s}_\mathbf{y}, \mathbf{y}_j) \\
&\qquad + \sum_{i=1}^{p}\sum_{j=1}^{p} k_\mathbf{y}(\mathbf{s}_\mathbf{y}, \mathbf{y}_i)\mathbf{K}^{-1}_{y,ij}k_\mathbf{x}(\mathbf{s}_\mathbf{x}, \mathbf{x}_j)) \\
&= -\frac{1}{2}(\sum_{i=1}^{p}\sum_{j=1}^{p} k_\mathbf{x}(\mathbf{s}_\mathbf{x}, \mathbf{x}_i)(\mathbf{K}^{-1}_{x,ij} + \mathbf{K}^{-1}_{y,ij})k_\mathbf{y}(\mathbf{s}_\mathbf{y}, \mathbf{y}_j))
\end{aligned}$$

With $\mathbf{K}_x = \mathbf{X}\mathbf{X}^{\mathrm{T}}$ and $\mathbf{K}_y = \mathbf{Y}\mathbf{Y}^{\mathrm{T}}$ one recovers standard pseudo-inverse learning energy function presented in the previous section.

Now we need to kernelize the inference process completely:

$$\begin{aligned}
s_\mathbf{y}(t+1) &= \underset{\mathbf{y}\in\{-1,+1\}^n}{\arg\min}\ E(\mathbf{s}_\mathbf{x}(t), \mathbf{y}) = \mathrm{sgn}\left(-\frac{\partial E}{\partial \mathbf{y}}\right) \\
&= \mathrm{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p} k_\mathbf{x}(\mathbf{s}_\mathbf{x}, \mathbf{x}_i)(\mathbf{K}^{-1}_{x,ij} + \mathbf{K}^{-1}_{y,ij})\frac{\partial k_\mathbf{y}}{\partial \mathbf{y}}(\mathbf{y}, \mathbf{y}_j)\right)
\end{aligned}$$

And analogously for $\mathbf{x}$ we would arrive to:

$$s_{\mathbf{x}}(t+1) = \operatorname{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\frac{\partial k_{\mathbf{x}}}{\partial \mathbf{x}}(\mathbf{x},\mathbf{x}_i)(\mathbf{K}_{x,ij}^{-1}+\mathbf{K}_{y,ij}^{-1})k_{\mathbf{y}}(\mathbf{s}_y,\mathbf{y}_j)\right)$$

It would be natural to compute the derivatives $\frac{\partial k_{\mathbf{x}}}{\partial \mathbf{x}}$ and $\frac{\partial k_{\mathbf{y}}}{\partial \mathbf{y}}$ from the previous $s_{\mathbf{x}}$ and $s_{\mathbf{y}}$, but in some preliminary experiments we have obtained better results using $\mathbf{0}$, a *neutral point* in the bipolar vectors space. For example, using an RBF kernel for both $k_{\mathbf{x}}$ and $k_{\mathbf{y}}$ we would obtain:

$$\begin{aligned}
s_{\mathbf{y}}(t+1) &= \operatorname{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}},\mathbf{x}_i)(\mathbf{K}_{x,ij}^{-1}+\mathbf{K}_{y,ij}^{-1})\frac{\partial k_{\mathbf{y}}}{\partial \mathbf{y}}(\mathbf{0},\mathbf{y}_j)\right)\\
&= \operatorname{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\exp(-\gamma\left\|\mathbf{s}_{\mathbf{x}}-\mathbf{x}_i\right\|^2)(\mathbf{K}_{x,ij}^{-1}+\mathbf{K}_{y,ij}^{-1})2\exp(-\gamma\left\|\mathbf{y}_j\right\|^2)\mathbf{y}_j\right)\\
&= \operatorname{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\exp(-\gamma\left\|\mathbf{s}_{\mathbf{x}}-\mathbf{x}_i\right\|^2)(\mathbf{K}_{x,ij}^{-1}+\mathbf{K}_{y,ij}^{-1})\mathbf{y}_j\right)\\
s_{\mathbf{x}}(t+1) &= \operatorname{sgn}\left(\sum_{i=1}^{p}\sum_{j=1}^{p}\mathbf{x}_i(\mathbf{K}_{x,ij}^{-1}+\mathbf{K}_{y,ij}^{-1})\exp(-\gamma\left\|\mathbf{s}_{\mathbf{y}}-\mathbf{y}_j\right\|^2)\right)
\end{aligned}$$

Finally, we have to consider that although this method is much more general we now have **quadratic complexity** with the number of memories which can be a computational bottleneck even for moderate size problems.

7 Sparse Kernelized Associative Memories

As mentioned, the main cons of KAMs are the need to store all memories and the need to iterate through all of them in the inference process. It is natural then to try to reduce the memories without sacrificing much performance through methods that profit from the redundancy with sparse KAMs. We will use the KBAM as the *base AM* to kernelize because of the computational requirements of its companions KOLAM/KRAM.

A **Stochastic KAM** only uses a random subset of the stored memories in each iteration speeding up the inference process. This simple method allows to reduce the number of computations in a inference step to a fraction r, the ratio of memories that will be used in that iteration.

The resulting equations will then be:

$$E(\mathbf{x}, \mathbf{y}) = -\sum_{i=1}^{\lceil p \cdot r \rceil} k_{\mathbf{x}}(\mathbf{x}, \mathbf{x}_i) k_{\mathbf{y}}(\mathbf{y}, \mathbf{y}_i)$$

$$\mathbf{s}_{\mathbf{y}}(t+1) = \operatorname{sgn}\left(\sum_{i=1}^{\lceil p \cdot r \rceil} k_{\mathbf{x}}(\mathbf{s}_{\mathbf{x}}(t), \mathbf{x}_i) \frac{\partial k_{\mathbf{y}}}{\partial \mathbf{s}_{\mathbf{y}}}(\mathbf{0}, \mathbf{y}_i) \right)$$

$$\mathbf{s}_{\mathbf{x}}(t+1) = \operatorname{sgn}\left(\sum_{i=1}^{\lceil p \cdot r \rceil} \frac{\partial k_{\mathbf{x}}}{\partial \mathbf{s}_{\mathbf{x}}}(\mathbf{0}, \mathbf{x}_i) k_{\mathbf{y}}(\mathbf{s}_{\mathbf{y}}(t), \mathbf{y}_i) \right)$$

The optimal value for the hyperparameter r will depend on data and it will be a trade-off between accuracy and inference speed. We will still need to store all the training memories as the random subset changes in each step.

8 Experiments

In this section we will explain and show the results of a series of tests that we have done to the memories mentioned through the thesis. We believe that this is one of the most important parts of the project as it will give perspective to the capacity of the AMs and how do the new variations do.

We will compare the different AMs efficiency and efficacy with controlled tests or using established benchmarks datasets found in previous literature in **Memory Capacity**, **Noise Robustness** and **Classification**. Although the ultimate goal is this comparison, every experiment has a concrete purpose that we will explain in its respective section.

The tested memories (categorized by *kind*) will be:

- **Classical** LAM, OLAM, BAM and RAM.
- **Kernelized** KLAM, KOLAM, KBAM and KRAM (RBF kernel).
- **Sparse** SKAM.
- **Special** PBAM (BAM structure with pseudoinverse training).

8.1 Memory Capacity

In this experiment we will try to see how many patterns association can the AM recall in an ideal situation, without distortion nor ambiguous memories. We think that **capacity** is one of the most relevant metrics about associative memories and that is why this first experiment is included.

In each realization of the experiment we will train all the memories with an increasing set of bipolar vectors associations and we will see how many of them are recalled correctly afterwards. Each experiment will be defined by the size of the inputs and the outputs, $\mathbf{m}$, and the number of memories $\mathbf{M}$.

All the realizations have been repeated 100 times for statistical validity.

8.1.1 Theoretical Bound

As there is no distortion involved nor ambiguous memories (the patterns are one-to-one) a *perfect algorithm* would return the correct output with a **100%** effectiveness, it would always correctly recall the correct association.

8.1.2 Results

We have tested all the memories for different M values using $m = 16$, Figure 2, and $m = 32$, Figure 3. We have chosen relatively small values in order to allow us to repeat the experiments many times without a high computational cost. Also, we have separated the normal and the kernelized methods because of their different capabilities and to make the visualization clearer.

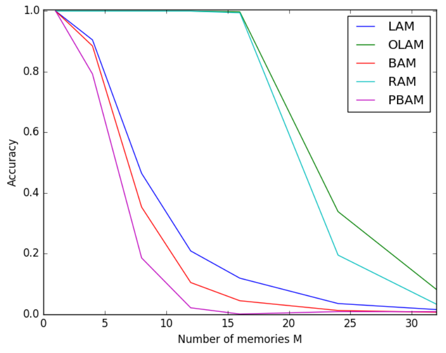
The first thing to notice is the *perfection* that kernelized memories exhibit in these settings, with a 100% effectiveness. This is something that happens for all the different tested m's as well as with much bigger M's that the shown in the graphs. This tells us that the requirement to have to store all memories is not in vain, we are assured to have perfect recall when the original patterns are presented again without any kind of distortion.

The second thing to notice is the significant performance difference between pseudoinverse-based methods and the Hebbian ones in the non-kernelized case. For both experiments, $m = 16$ and $m = 32$, we consistently see that

Hebbian start making mistakes with more than 4 memories while pseudoinverse methods recall acurately until $m = M$. This is coherent with the approximate capacity bounds studied in the original BAM paper [7].

Also it is patent that the PBAM, the *Frankenstein*-like memory described at **Bidirectional Associative Memory** has the worst results. As we predicted we can not combine enforcing the symmetry constraint and using a pseudoinverse training method, it just does not work.

Finally, we can also observe that recurrent associative memories perform a little bit worse than their direct-inference counterparts in the majority of m and M cases. This is an unintuitive result, we would normally think that the recurrence generalization is always positive, and it will be explained in the **Discussion** section jointly with the other experiments.



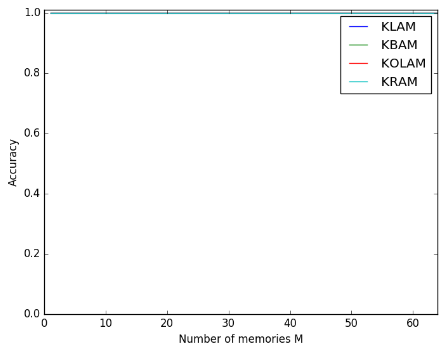
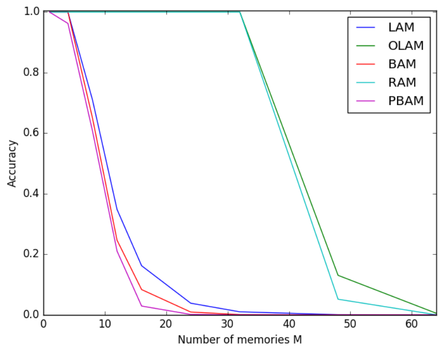


Figure 2: Memory Capacity experiment, $m = 16$.
On the **right** figure all methods have 100% accuracy for every M value.



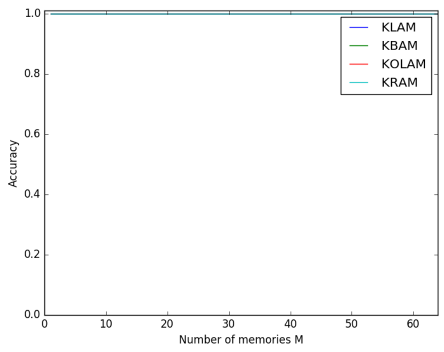


Figure 3: Memory Capacity experiment, $m = 32$.
On the **right** figure all methods have 100% accuracy for every M value.

8.2 Noise Robustness

This experiment is a modification of the previous one. After the initial training we do not present the original input patterns to the AM, we present distorted versions of them, some of their components have been flipped.

In this experiment then we will try to determine how robust are the memories to small distortions in their patterns as well as the capacity of the memory. We think that **noise robustness** is another essential metric of an associative memory for many reasons. Historically this was one of their firsts uses [7] and in our opinion this relatively *small* generalization task hints the associative memory true potential and capabilities.

Each realization of the experiment will be characterized again by the input/output size **m**, the # of memories **M** and the # of corrupted bits **N**.

All the realizations have been repeated 100 times for statistical validity.

8.2.1 Theoretical Bound

In this case we need to be a bit more careful as actually smallish distortions can really make some of the memories **unrecoverable**. If too many components are flipped the corrupted memory may actually be more similar to other patterns in the memory and correct guessing just be a matter of luck.

It is very interesting then to try to find this maximum in order to see how close or far stand the analyzed memories. Unfortunately, we have not arrived to a closed-expression formula of this bound but below we have an example of how we could calculate it for each case.

Example (m = 8, M = 2, N = 1)

We have to take into account the cases where the corrupted memory may be confused for another memory and compute their probability. In this example:

- The 2 memories are one bit away, $p_1 = \frac{8}{2^8-1} \approx 2^{-5}$, the noise affects that bit, $q_1 = \frac{1}{8} = 2^{-3}$, and the probability of error, $r_1 = \frac{1}{1} = 1$.
- The 2 memories are two bits away, $p_2 = \frac{8*7}{2^8-1} \approx 0.22$, the noise affects

one of those bits, $q_2 = \frac{2}{8} = 2^{-2}$, and the probability of error, $r_2 = 2^{-1}$

Now we just have to add them as $1 - (p_1 * q_1 * r_1 + p_2 * q_2 * r_2) \approx \mathbf{0.97}$

In practice we have just approximated and simulated this bound.

8.2.2 Results

In order to test the noise robustness in different situations we have done 3 experiments, Figure 4 for an almost empty memory ($M = 2$) and Figure 5 and Figure 6 for fuller ones ($M = 8$ and $M = 16$). In all cases we have used $m = 16$, an smallish memory size to speed up experimentation.

We can notice again in all the experiment realizations that kernelization is a huge improvement, there is a big performance difference comparing with their normal counterparts. And not only that, we can see that they are all actually quite close to the simulated theoretical bound proving again their pattern association capabilities.

We can also observe that the best performers in these experiments are our novel memories **KOLAM** and **KRAM**. The latter, the recurrent version of the former, is actually better in some realizations and we can consider it the real *winner*, the most noise-robust associative memory.

Another interesting insight is that the relative performance of the different associative memories depends on how full is the memory. We can see that recurrent memories do actually better than their non-recurrent counterparts on more noisy situations and worse in less noisy ones. This justifies the effects noted on the previous experiments where there was no noise and non-recurrent memories performed well.

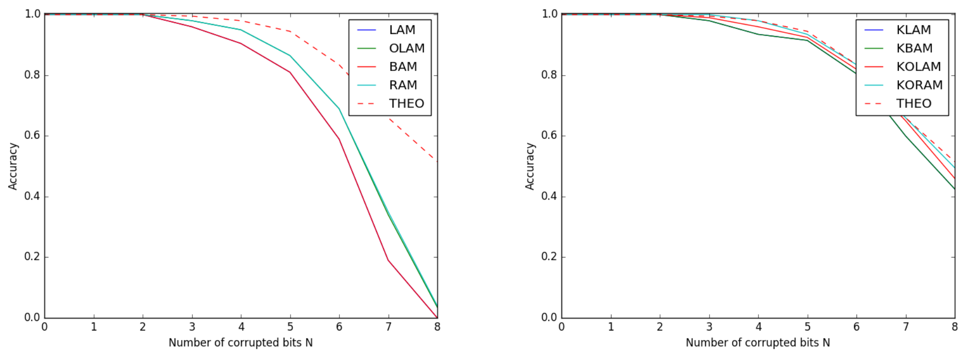


Figure 4: Noise Robustness experiment, $M = 2$

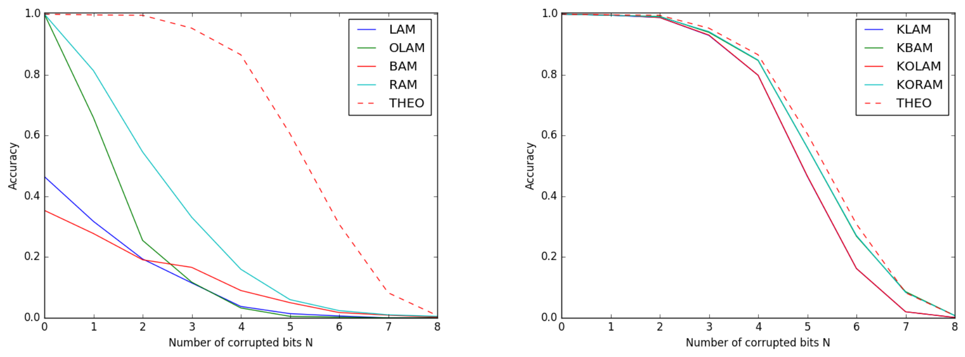


Figure 5: Noise Robustness experiment, $M = 8$

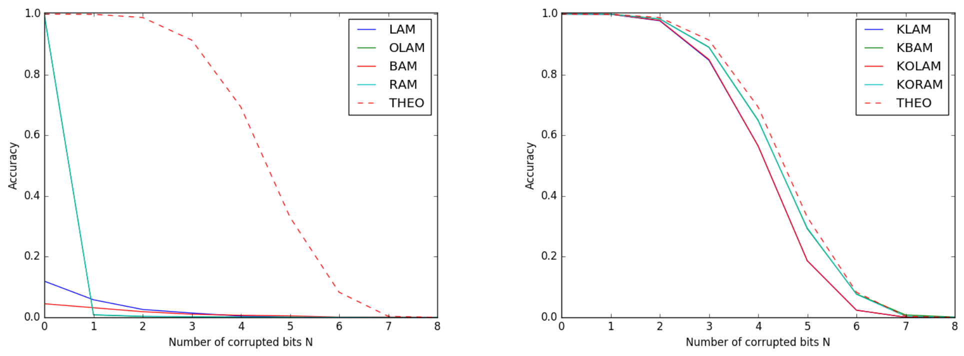


Figure 6: Noise Robustness experiment, $M = 16$

8.3 Classification

In this experiment we will show that AM's can also do classification with success. We have chosen the **MNIST** dataset, a large database of 60000 handwritten digits. The task will be to associate the images, that after being binarized can be seen as bipolar vectors, with the number they represent.

We think that this experiment will allow us to check how the associative memories fare in a real dataset, not synthetic like in the previous ones. Also, as it is not a injective dataset (many different handwritten numbers go to the same *category*), we can try the novel sparse memory SKAM that takes advantage of the redundancy present in the data to speed-up the inference process ideally without losing much accuracy.

We will not test recurrent associative memories as they make no sense in the classification context. Once the memory have classified a pattern, for example image X is 4, it will not change its opinion for no reason.

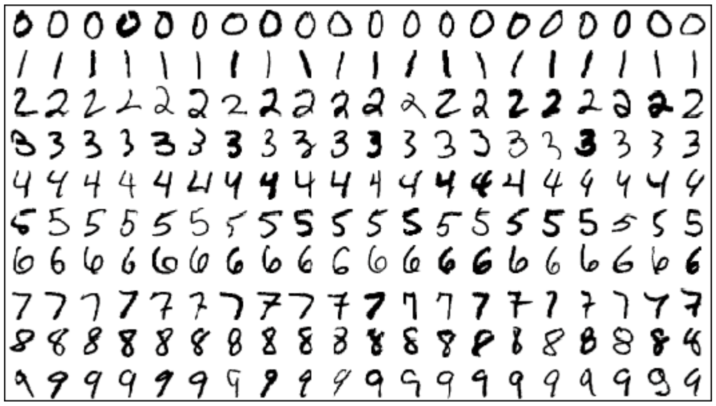


Figure 7: MNIST binarized handwritten digits examples

Memory	Precision (%)	Training Time	Inference Time (ms)
LAM	0.0	55 ms	42.6
OLAM	67.1	15.12 s	45.9
KLAM	96.9	-	590
SKAM ($r = 0.5$)	96.2	-	300
SKAM ($r = 0.1$)	93.6	-	61.5

Table 1: Classification experiment results

8.3.1 Results

In Table 1 we have a summary of the experiment obtained results. The kernelized memories do not have *Training Time* because they do not have a defined training process, they directly use in inference the training memories.

The first interesting results is the 0% accuracy of the simple LAM. This is because the memory never predicts any number, it gets overload with information in training and makes the minor cost option, predicting *no number*.

We can also notice the tremendous performance gain that kernelization provides, from a 67.1% to almost all of them. This is consistent with the synthetic experiments results showing that kernelization is also positive with real data. Probably using our new KOLAM we would have obtained even better results but because of the matrix kernel generation computational it is completely impossible to run in this dataset.

Finally, our *sparse* associative memory has worked as we expected, having very good results and reducing the recall time by an r factor. With $r = 0.5$ we are almost twice as fast with almost the same performance and with $r = 0.1$ we are almost ten times faster keeping 93.6% accuracy.

9 Discussion

After the experimentation are done and their results are stated it is important to analyze them and justify the observations with arguments. Explaining and understanding these experiments results in an essential part of the project.

We have extracted the most significant facts and explained them in detail in the next subsections without a relevant order.

9.1 Capacity Increase by Kernelization

As noted in previous work, when we kernelize the associative memory we observe in all experiments a very significant capacity increase. It can also be seen that this difference depends on the used kernel in a consistent way.

We can explain why this happens having in mind the next facts:

- Kernelization transforms implicitly $\mathbf{x}$ to $\phi_x(\mathbf{x})$ and $\mathbf{y}$ to $\phi_y(\mathbf{y})$, vectors in a higher-dimensional space of S_x and S_y dimension (potentially infinite).
- The resulting size of $\mathbf{W}$ after kernelization is $S_x \times S_y$.
- The capacity of an AM depends on the degrees of freedom of the design matrix $\mathbf{W}$ (m), the bigger it is the more memories it can store.

This can also explain why the RBF kernel has in general a better performance than the polynomial one, as the space dimension of the former is larger.

9.2 Pseudoinverse Learning usefulness

One of the objectives of this project was to develop a new kind of associative memory that we called **KOLAM** and **KRAM**. We have also discovered in the experiments that they provide an small but consistent better performance than kernelized methods based on Hebbian training.

As another of our objectives is to take into account efficiency we think that it is important to analyze and explain the observed results.

9.2.1 Redundancy with kernelization

We can observe that in all experiments the performance gained using pseudoinverse methods is smaller in kernelized memories than in normal ones.

A possible explanation of this fact is the redundancy of kernelizing and pseudoinverse-training benefits, because as proved in [11], both methods orthogonalize the input patterns, one implicitly and the other by the design.

We can also see that if the kernelization *almost* orthogonalizes the patterns, Hebbian and pseudoinverse training are *almost* equivalent:

$$\phi(\mathbf{X})^{+}\mathbf{Y} = \phi(\mathbf{X})^{\mathrm{T}}(\phi(\mathbf{X})\phi(\mathbf{X})^{\mathrm{T}})^{-1}\mathbf{Y} \approx \phi(\mathbf{X})^{\mathrm{T}}(\mathbf{I})^{-1}\mathbf{Y} = \phi(\mathbf{X})^{\mathrm{T}}\mathbf{Y} \tag{4}$$

9.2.2 Complexity

One of the negative parts of these new memories is the huge computational cost they require even preventing them to complete the MNIST experiment. The computation of the energy function and an inference step needs to go through all the combinations x_i-y_j, a quadratic complexity in the number of stored patterns which is normally a very big number.

The most important fact is that this *slowness* is not at training, that only has to be done once, it is at the inference phase, we have to do it every time we want to recall a memory...

9.3 Recurrence Effectiveness

One of the most interesting observations of the experiments is that recurrence is not always positive, sometimes simple linear versions have slightly better results. This happened specially on low noise situations.

We can theorize that this happens because in those *easy* situations the recurrence can induce to find further (but lower) local minima on the energy function when the real solution was the closest one.

As we have seen with the disastrous PBAM results we can not successfully combine symmetry constraints with pseudoinverse training.

9.4 Sparsity Impact

In **Classification** experiment we tested our new memory **SKAM** and compared them to different non-recurrent structures. Using $r = 0.5$ we obtained a memory that was twice as fast losing less than 1% accuracy and with $r = 0.1$ we obtained a memory that had the same computational cost than a OLAM but with more than 25% accuracy.

We have to say two important things about the *sparsity* of our SKAM.

- **Storage Sparsity:** We are not doing complete sparsity in the sense that we have to store all memories independently of r, we are just ignoring them temporarily in some inference steps. In order to do real sparsity we would to analyze the dataset in training time (we propose it in the **Future Work** section).
- **Complexity:** We want to make clear that with these methods we are not tackling one of the most important problems, the quadratic complexity of kernelized pseudoinverse methods. A completely new idea would have to be used in order to make them computationally affordable in real datasets.

Despite this, we think that these results are very encouraging and that sparse associative memories may offer a good trade-off between accuracy and efficiency for many problems where just a moderate speedup is needed.

In the following three sections, **Work Plan**, **Budget** and **Sustainability**, we will analyze the project from a management point of view.

It is important to keep in mind that some of the treated topics, like costs estimation, are not real as this is just a bachelor's thesis with no budget assigned. But, doing this *simulation* of sorts, does help us to realize the amount of work devoted to the project and to analyze its structure and consequences.

10 Work Plan

In this section we will explain the project structure, the tasks in which it has been divided and the resources used in order to carry it out. Moreover, we will explain the action plan followed, how the tasks have been done, and different proposals of alternatives and solutions to considered obstacles.

This project officially started on September 12th of 2016 and has been delivered to the examiners' panel on January 17th of 2017.

10.1 Project Structure

In order to make a clear structure the project has been divided in 6 big blocks that partially overlap. We will explain first the general idea of each of them and their weight in the project.

10.1.1 Project Idea

The conception of the project idea and the first outline of the objectives was made before the registration of the TFG, in July 2016.

The topic was proposed by Lluís A. Belanche, the tutor. We wanted a research topic in the area of machine learning and he recently had guided a master thesis [12] that had proposed many new paths to be continued.

Although some brainstorming meetings were done we will exclude them from the other sections as they were out of the official project schedule.

10.1.2 Initial Stage

The initial stage goes from September 19th to October 24th of 2016.

In this stage we developed the project foundations, both technical and more management related. We tried to make clear the scope, objectives and create an rough initial planning from the very first moment in order to reduce risks and make the project firsts steps.

10.1.3 Development

The development of the project, mathematical and algorithmic, is the core of the project from September 30th to November 18th.

As we have explained in **Objectives** this project has not only one goal and this fact is reflected in the variety of tasks of this block.

Although this section comes theoretically after initial stage ending it actually started some weeks before.

10.1.4 Experimentation

As in almost all research projects experimentation is an essential part that takes quite a long time. We planed from October 10th to December 5th.

In this block I consider the experimentation related to the testing, to verify that the developed code is correct, as well as the design and execution of all the different experiment and applications shown in the thesis.

10.1.5 Documentation

One of the most forgotten parts in project planning but the imperative to do!

In this block I include internal documentation, intermediate results of failed methods, and the real documentation, the thesis and the code.

10.1.6 Extra

At the end of the project, from December 12th to January 17th, we decided to set an *extra time* to try new ideas not initially planned or end the final tasks if we were behind the planned schedule.

We finally had to use all of this time finishing up documentation and experiments as there were some miscalculations in some tasks resources estimation.

Task	**Description**	**Time**
Initial Stage	Initial assessment document tasks	78 h
Framework Structuring	Creation of the new software structure, flexible and general enough to implement the different algorithms.	18 h
Baselines implementation	Implementation of the non-novel algorithms to test the framework.	33 h
Pseudoinverse Methods	Implementation of all non-novel pseudoinverse training methods.	18 h
Heteroassociativity	Study and development of methods related with heteroassociativity.	66 h
KOLAM & KRAM	New kernelized associative memories programming and testing.	33 h
SKAM	Programming and testing of new sparse associative memory.	18 h
Experiments (I)	Experiments related to memory capacity and noise robustness.	18 h
Experiments (II)	Experiments related to classification and efficiency.	33 h
Thesis Draft	Creation of first thesis draft.	33 h
Extra	Unassigned time to finish the thesis and try new ideas.	93 h
Total		**441 h**

Table 2: Tasks

10.2 Resources

The project resources planning can be divided in **hardware** and **software**. We will not consider human resources as there really is just one *worker*.

10.2.1 Hardware

We have used a 1.7 GHz i5 processor laptop, where we have programmed and have written all the other documents, and a 2.66 GHz i5 desktop where all the tests and all the final experiments have been done.

The other specifications are not mentioned as they are not relevant.

10.2.2 Software

The programs used in the elaboration of the thesis are:

• **Atom**	Text editor used to write all the code.
• **gantter**	Cloud-based project scheduling tool.
• **GitHub**	Online project hosting using git.
• **Julia**	Programming language used for all programming tasks.
• **LaTeX**	Typesetting system used to write the documentation.
• **Texmaker**	Cross-platform LaTeX editor.
• **Ubuntu**	OS used in both systems during the project.

10.3 Action Plan

We decided to initially follow tasks of a duration of some weeks and subdivide them in one or two weeks chunks when we started them. During the project we found out that sometimes those estimations were not very accurate and had to be adjusted slightly delaying some other tasks.

In order to find bugs as soon as possible small tests were planned after every programming tasks and theoretical derivations were reviewed by the tutor.

The **Extra** block was initially planned to try new ideas if the project was on schedule and act as a safety margin if there were delays. It was finally used to finish some tasks and do some thesis drafts reviewing iterations.

11 Budget

In this section we will make some guesses and estimations of how much the project would have cost if it was a real business venture. We will break down costs, evaluate some of the possible risks and briefly talk about how do we would the project control management to avoid major evils.

11.1 Costs Estimation

We think that the typical cost breakdown by tasks is not well suited for this kind of project with algorithms development and experimentation. It is more interesting to talk directly about **human**, **software** and **hardware** costs.

11.1.1 Human Resources

The table below shows the considered human resources costs.

Role	Price per hour	Time	Cost
Project Manager	50.00 €	95 h	4750.00 €
Researcher	40.00 €	170 h	6800.00 €
Programmer	30.00 €	110 h	3300.00 €
Tester	20.00 €	66 h	1320.00 €
Total		**441 h**	**16170.00 €**

Table 3: Human resources costs

Manager In charge of project planning and management. Responsible of **Initial Stage** and writing definitive **Documentation**.

Researcher In charge of the project theoretical part. Responsible of the algorithms development and the experiments design.

Programmer In charge of algorithms and experiments programming and other low-level implementation details.

Tester In charge of doing the experiments, annotate and verify results as well as report issues and execution problems.

11.1.2 Software Resources

The table below shows the considered software resources costs.

Product	**Price**
Atom	0.00 €
gantter	0.00 €
GitHub	0.00 €
Julia	0.00 €
Texmaker	0.00 €
Ubuntu	0.00 €
Total	**0.00 €**

Table 4: Software resources costs

As all the used technologies are free we did not need to buy expensive licenses and we could use that saved money in other important areas.

11.1.3 Hardware Resources

The table below shows the considered hardware resources costs.

Product	**Price**	**Amortization**
Laptop	400.00 €	67.00 €
Desktop	700.00 €	117.00 €
Total		**184.00 €**

Table 5: Hardware resources costs

We have used a 3-year period in amortization calculations as the hardware is low-end, it has a short life.

Example: $400.00\text{ €} * \frac{0.5\ years}{3\ years} \approx 67.00\text{ €}$

11.1.4 Unforeseen Costs & Contingencies

All projects are exposed to numerous risks and the best way to deal with it is **preventing**. We have reflected about what could go wrong and we have decided that biggest predictable risks are:

Risk	Probability	Cost	Exposure
Experiment failure	10%	600.00 €	60.00 €
Method failure	10%	800.00 €	80.00 €
Re-planing	5%	500.00 €	25.00 €
Total			**165.00 €**

Table 6: Contingencies cost

We have created an especial budget item called **Contingency Plan**.

Moreover, as we want to be prudent and sensible, we have decided to add an extra 5% over the principal costs in order to avoid **Unforeseen Costs**.

11.1.5 Total Cost

In the table below we can see the summary of all the previous costs.

Item	Cost
Human Resources	16170.00 €
Software Resources	0.00 €
Hardware Resources	184.00 €
Contingency Plan	165.00 €
Unforeseen Costs	818.00 €
Total	**17337.00 €**

Table 7: Total cost

We can see that the biggest cost by large is the **Human Resources** item it is a project with low hardware and software requirements.

11.2 Control Management

In order to control the project status **weekly checkups** will track the number of real work hours and the task progress. In this way the deviations will be quickly detected and corrected if they are significant enough.

For example, to cope with a delay in some task we would first prioritize it and, if deemed necessary, use Extra resources as a last resort.

12 Sustainability

Nowadays the sustainable development of projects is of vital importance. As a society we have realized that we have to take care of the resources we have since they are scarce and we may run out of them in a future.

In order to analyze this complex topic we have decided to divide it and analyze its economic, social and environmental dimensions separately.

12.1 Economic

A thorough economic evaluation has been done at **Budget** and the project utility and decisions have been assessed objectively.

We think that as it is a research project, we are not competing against rivals, the schedule does not have to be extreme. Despite this, we have given importance to task planning and costs estimation so that we do not waste resources.

The developed methods in this thesis may be used in more practical applications in a future but we have not considered them in the analysis as it is impossible to predict them and we do not feel responsible for them.

12.2 Social

We consider the social impact of this project negligible, we are only developing learning algorithms for associative memories.

We do not harm nor help any social collective and, as we have previously explained, we do not consider possible future applications.

12.3 Environmental

The only environmental resource used in this project is the energy consumed by the computers as it is a software development project without peripherals involved. We could also consider the energy consumed in the desktop and laptop manufacturing but that would only over-complicate the analysis.

We can make a simple environmental impact estimation considering a time dedication, 441h, and the consumption of both computers, 250 W.

$$E = 441\ h * 250\ W \approx 110\ kWh$$

We can also compute the estimated pollution emitted using 0.5 $\frac{kg\ CO_2}{kWh}$ as ref.

$$C = E * 0.5 \approx 55\ kg\ CO_2$$

12.4 Sustainability Matrix

After the explanations of the previous subsections we can show a very well-known metric of sustainability, the **Sustainability Matrix**. All the figures have been obtained taking into account the previous explanations.

	Economic	Social	Environmental
Development	9	7	7
Exploitation	14	16	14
Risks	-4	0	-4
		59	

Table 8: Sustainability Matrix

13 Future Work

There were many interesting ideas that we did not have time to fully develop and include for this thesis because of our limited time and resources. Here we will explain the ones that we thing that we consider more promising with hopes that they are continued in another projects.

One of the most interesting paths to pursue is about the sparse AM's. We have only proposed a simple stochastic variant (SKAM) but we think that many other approaches are possible. For example, we tried to create a Weighted KAM, a memory that analyzing the dataset gave a weight to every pattern to learn and focusing on creating 0's (effectively discarding them). This would be completely sparse, we would not even have to store the discarded memories. We could not make it work but it is a promising idea.

Another thing that we have not studied in the project is the convergence of the kernelized associative memories. It is not obvious which properties of their non-kernelized counterparts remain in those memories as the structure is severely changed.

Finally, we also encourage to try these memories on different applications as we have shown their classification capabilities and flexibility.

14 Conclusions

In this thesis we have developed a new learning framework for associative memories and aside from implementing the most fundamental past methods we have proposed two new ones. The first one, KOLAM/KRAM, combines kernelization and pseudoinverse training something novel in an heteroassociative context. The second one, the Stochastic KAM, is a first attempt to speed-up recall in kernelized associative memories and increase their efficiency, another of the main goals of the project.

We have done three experiments to find out how these new methods compared with the previous ones obtaining some key insights. First, the most complete method KRAM had the best in all situations getting very near the theoretical bound in the synthetic experiment. Second, SKAM proved to provide a very a useful trade-off between accuracy and speed reducing recall time considerably while having highly competitive results. Third, we need to find out a method to make kernelized pseudoinverse methods computationally affordable because of their intrinsic quadratic complexity.

Last but not least, we have also studied and documented the management part, the used methods in the project, a budget estimation and a sustainability study. We have learned that it is essential to estimate correctly the project tasks duration and that it is highly recommended to have contingency plans like *a priori* unassigned days before deadlines to avoid failure.

References

[1] Maria Elena Acevedo-Mosqueda, Cornelio Yáñez-Márquez, and Marco Antonio Acevedo-Mosqueda. Bidirectional associative memories: Different approaches. *ACM Computing Surveys (CSUR)*, 45(2):18, 2013.

[2] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *arXiv preprint arXiv:1411.1607*, 2014.

[3] B Caputo and H Niemann. From markov random fields to associative memories and back: Spin glass markov random fields. *SCTV2001*, 98:101–102, 2001.

[4] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory.* Psychology Press, 1949.

[5] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[6] Y-J Jeng, C-C Yeh, and TD Chiueh. Exponential bidirectional associative memories. *Electronics Letters*, 26(11):717–718, 1990.

[7] Bart Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):49–60, 1988.

[8] Yann LeCun, Sumit Chopra, Raia Hadsell, M Ranzato, and F Huang. A tutorial on energy-based learning. *Predicting structured data*, 1:0, 2006.

[9] G Mathai and BR Upadhyaya. Performance analysis and application of the bidirectional associative memory to industrial spectral signatures. In *Neural Networks, 1989. IJCNN., International Joint Conference on*, pages 33–37. IEEE, 1989.

[10] Dimitri Nowicki and Hava Siegelmann. Flexible kernel memory. *PloS one*, 5(6):e10955, 2010.

[11] Raúl Rojas. *Neural networks: a systematic introduction.* Springer Science & Business Media, 2013.

[12] Matthew Saltz. A study into kernelized associative memories. 2015.

[13] Jürgen Schmidhuber. Deep Learning in neural networks: An overview, 2015.

[14] Haihong Zhang, Bailing Zhang, Weimin Huang, and Qi Tian. Gabor wavelet associative memory for face recognition. *IEEE Transactions on Neural Networks*, 16(1):275–278, 2005.