Implementation and study of a deep and wide Kernel Machine

Bachelor Degree Final Project

Barcelona School of Informatics,
Technical University of Catalonia,
Barcelona Tech

June 23, 2015

Director:
BELANCHE MUÑOZ, Lluís

Author:
SERRA ALZA, Xavier
Abstract

This project consisted in the implementation and test of a particular kind of a Deep Kernel Machine. This kind of program belongs to the field of Machine Learning, and, therefore, is used to make predictions based in a previous process of learning. In this project, these predictions consisted in correctly classifying handwritten digits, ranging from 0 to 9. Contrary to most computer vision approaches, this method does not extract special features from the image but, instead, uses raw images to learn on its own.

The method used, called Deep Kernel method, uses a similarity function between known examples and the new images to determine the value of the latter. This similarity function is called arc-cosine kernel, and it is thoroughly explained later in this document.

The images of digits used in the project have been obtained from the MNIST dataset, which contains 70000 examples of handwritten digits. In order to make the predictions more realistic, these images come from different sources.

The prediction results obtained by the implemented method show near state-of-art quality, and have been obtained in a fraction of the computational time usually associated with other more standard Deep Learning methods, with a minimum error of 1.59. Moreover, the process required to obtain them is not excessively hard nor time consuming.
Acknowledgment

First of all, I would like to thanks the director of the project, the doctor Lluís Belanche, for all the theoretical advisement, the comprehension, and last but not least, the encouragement necessary to finish the project. Without him, this project would not exist.

I also wish to thank my girlfriend Cristina, who has managed to cheer me up even in the most difficult moments, and who has understood me for all the hours spent in the project.

I would also like to thank my parents and my little brother, for their tolerance with all the hours I have spent locked in my room, and for the interest they have shown in the project. This motivation has helped me to keep carrying on.

And I also thank my friends, for their peculiar way of supporting me, which has helped me keeping a good mood during all the project.

Finally, I wish to thank the Computer Science department for the access provided to their computing cluster, which has allowed me to obtain more and better results than I could have on my own.
## Contents

1 Introduction ........................................ 8
   1.1 Context ........................................... 8
       1.1.1 Origins of Machine Learning ................. 8
       1.1.2 Appearance of Deep Learning ............... 9
   1.2 State of Art ...................................... 12
   1.3 Motivation ....................................... 12
   1.4 Project Description ............................... 13
       1.4.1 Goals of the project ......................... 13
       1.4.2 Range of the project ......................... 14
       1.4.3 Document structure ......................... 14

2 Theoretical Background ................................ 16
   2.1 Machine Learning .................................. 16
   2.2 Kernel Methods ................................... 18
       2.2.1 Linear support vector machines ............. 18
       2.2.2 Nonlinear SVMs ............................... 20
       2.2.3 Existent Kernels ............................. 22
   2.3 Deep Learning ................................... 23
       2.3.1 Kinds of Deep Learning ...................... 24
           2.3.1.1 Supervised Learning .................. 24
           2.3.1.2 Unsupervised Learning ............. 24
       2.3.2 Most common Deep Learning algorithms ...... 25
   2.4 Deep Kernel Methods ............................. 27
3 Project Management

3.1 Obstacles and risks

3.2 Following of the project

3.3 Schedule

3.3.1 Dependency between tasks

3.3.2 Deviations and Corrections

3.3.3 Changes on the schedule

3.3.3.1 Consequences and justification

3.4 Sustainability

3.4.1 Economical aspect

3.4.2 Social aspect

3.4.3 Environmental aspect

4 Project Development

4.1 Programming Language Used

4.2 SVM-Solver

4.2.1 libsvm

4.2.1.1 Adaptation of libsvm

4.3 Arc-cosine Kernel
1 Introduction

1.1 Context

1.1.1 Origins of Machine Learning

Ever since its very beginning, one of the main goals of computer science has been to be able to create a machine capable of mimicking human behaviour. Much research has been made in this area, but the goal of replicating a whole human intelligence was considered unfeasible long ago, at least at the moment. Instead, the problem of the human intelligence has been divided into many sub-problems which have been, and currently are, studied. Some of these areas are: computer vision, natural language recognition, planning, etc. All of these problems have in common the need for a decision making process, whether it is to decide if a certain image does or not contain a human being, or to determine the mood of the writer of a sentence. The making of decisions, therefore, has become of great interest in the field of computer science, as it affects, directly or not, many others. However, the traditional approaches proved insufficient, as having to manually determine the answer to every action, even though it may work in limited environments, would not be enough in a real situation.

The main goal of statistical Machine Learning [24], instead of focusing in teaching programmes how to act, is actually teaching them how to learn, allowing them to evolve on their own. This method results in the program learning patterns, or models, which can be used in many
Introduction

Implementation and study of a deep and wide Kernel Machine

different situations, allowing for more intelligent, or, at the very least, flexible, decisions. Moreover, Machine Learning is not only used to try and imitate human decisions, but to outperform it, being a computer able as it is to analyze enormous quantities of data. By doing this, computers may find relations invisible to the human eye, with applications varying from the financial world to cancer analysis. That being said, even though computers may actually do better than humans in certain tasks, they do have limitations, like, for example, recognizing handwriting. A task so seemingly easy has proven way more difficult for machines than it is to us, with most traditional methods obtaining unacceptable error rates. The same situation has also happened in other problems with extremely complex data, such as image or speech recognition, in which even the most state-of-art models proved unable to solve them.

1.1.2 Appearance of Deep Learning

One of the currently most common types of neural networks is the Restricted Boltzmann Machine [15]. This kind of neural network can be used as a classifier, but also as a way to reduce the dimensionality of the data. Since some years ago, RBM have been used to design multi-layer neural networks, producing as a result one of the most well-known algorithms of Deep Learning[16, 2]: the Deep Belief Network[17]. This technique consists in providing the input data to a RBM, the output of which is supplied to another RBM with a smaller number of neurons. A graphical representation of this can be found further in the Fig 1. This
Introduction

Implementation and study of a deep and wide Kernel Machine process can be repeated as many times as necessary until the desired number of layers is achieved. As every layer has an smaller number of neurons than the previous layer, data is compressed more and more with an increasing number of layers (more depth).

![Typical Deep Learning Scheme](image)

Figure 1: figure

Typical Deep Learning Scheme, with multiple layers [4]

This rather new branch of Machine Learning has provided new methods to attack such difficult problems, being able to obtain much better results than the previous ones [30]. This opened new options in areas in which researchers were stuck. That being said, Deep Learning was not a general solution for all Machine Learning problems, due to two main factors:

- Precision: Despite obtaining very good results in certain problems, it does not perform so well in other simpler problems, in which traditional methods are still the state of art.

- Execution time: The main drawback of Deep Learning is that the time required to train it is extremely high comparing with other
Introduction

As a result, Deep Learning became a highly useful method in certain problems which required being able to represent complex patterns, whereas traditional methods continued being the best option in many other areas. These traditional methods range from the relatively simple logistic regression to more complex neural networks, existing many others and multiple variations for every of them [22].

As seen, the notion of Deep Learning has been traditionally associated with neural networks, and most of the work related with it has used them. However, recently there has been the proposal of trying to apply Deep Learning to another method, which are the kernels. Kernels, thoroughly explained in 2.2, when applied to Machine Learning are exceptionally good at finding patterns in data, but up until recently they had not been combined with Deep Learning until the article Kernel Methods for Deep Learning [1], of Lawrence K. Saul and Youngming Cho. In this article, both researchers proposed a way of doing precisely this, and the results they had obtained by doing so. And it is about this article in which this project is based.
1.2 State of Art

Machine Learning currently raises a lot of interest. The main reason for this is the massive amount of data daily produced in the world of modern computers. This data is virtually impossible to manually analyze, and, therefore, its analysis is delegated in most cases to Machine Learning programs. On the other hand, Deep Learning is a much more recent field, meaning that there is much work to be done in this area, and even now it is no fully understood[23]. Finally, the usage of kernel methods along Deep Learning is even more recent than Deep Learning. Actually, the research done in this area is considerably scarce.

1.3 Motivation

Deep Learning is a comparatively recent area but holds a lot of potential. Additionally, as there is not so many people working in it, it is even more appealing. Finally, combining it with kernel methods2.2 is even less common, which is the one of the main motivations behind this project. On the other hand, the other decisive factor was the fact that the the results obtained by professors Cho and Saul in some cases were close to the state-of-art of other Deep Learning methods but required much less time.

The motivation here is being able to fully understand how the model described works, and to obtain a working program capable of using it. If possible, it would be interesting to introduce some modification in it.
1.4 Project Description

This project focuses in Deep Kernel methods, a rather new branch of an already recent field, as is Deep Learning. Because of this, it could be said that it belongs to an innovative field. However, truth is, this project is of bachelor degree and, therefore, it can not try to compete with real state-of-art applications. Thus, its goals are modest, aimed to the academical growth of the student in this field rather than making a great contribution to it. That being said, the project will also try to innovate as far as possible, taking into account the time constraints.

1.4.1 Goals of the project

In this section there are the goals of the project properly formalized. They are ranked approximately in order of relevance:

1. The student aims to develop a functional Deep Kernel machine, using for this the article *Kernel Methods for Deep Learning* [1]. The goal here is to fully implement the methods described in the article.

2. In order to analyze the performance of the method obtained, it will be tested in a set of data called *MNIST*, composed of images of handwritten digits.

3. If possible, this project will try to further develop the methods described in the already mentioned article, adding some modification. The goal here is not necessarily to improve the method
Introduction

Implementation and study of a deep and wide Kernel Machine

itself, but to try if further enhancements are possible.

As a side effect of this, the student will also get in touch with Deep Learning theory, which, even though not a goal in itself, will be necessary to understand better Deep Kernel methods. This will allow the student to familiarize himself with the understanding, development and use of Deep Kernel methods.

1.4.2 Range of the project

As it has already been mentioned, this project is not an actual research one, but, instead, a more educational based one. Therefore, its range is limited in the sense that:

- This project does not aim to develop a brand new Deep Kernel machine, but, instead, replicate the one designed by Cho & Saul [1], with the possibility of introducing certain modifications.

- This project does not aim to compete either in performance nor, especially, in computation time with state-of-art results.

1.4.3 Document structure

This rest of this document is divided into 5 sections:

1. *Theoretical Background*: In this section there is a briefing of the theory behind this project. It does not aim at providing a thorough explanation of Machine Learning, and, instead, focuses in Support Vector Machines[11] and arc-cosine kernels[1] which are
Introduction

the main point of this project. For further reference in the Machine Learning area, consult [22, 24].

2. *Project Management*: Contains the documentation all the necessary work previous to the beginning of the project, such as an schedule or an study of the sustainability of the project.

3. *Project Development*: Explains how the project developed, the difficulties faced, and the decisions made in it.

4. *Experiments*: This section contains the results of the experiments done with the method implemented, as well as an explanation of them.

5. *Conclusions*: This final section contains the conclusions extracted from the project, including the analysis of the work done and the evaluation of the goals of the project.
2 Theoretical Background

2.1 Machine Learning

In this section there is an intuition about Machine Learning [25]. Basically, Machine Learning programs improve their performance at some task through experience. More precisely:

Definition: A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.

For example, a computer program learning how to classify e-mails as spam or not spam, could be said to learn from the experience $E$ of watching this classification done by a human, with respect to the task $T$ of classifying emails, measuring the performance $P$ as the fraction of emails correctly classified as spam or not spam. In order to have a well-defined learning problem, the correct identification of these three tasks is usually essential: the class of task, the measure of performance to be improved, and the source of experience.

Some of the most well known uses of Machine Learning are classification, regression and clustering.
Theoretical Background

Classification  As its own name suggests, it consists in trying to classify each new instance in a certain category. It can be either a binary classification, for example to represent true or false, or with as many categories as needed. This kind of problems are mainly solved in a supervised way, providing the category of every instance.

EXAMPLE: A typical example of this kind of problems could be determining if a certain patient in a hospital has a malign or benign kind of tumor, which would the the two categories.

Regression  Regression problems are the ones that require a continuous quantity as a result, as opposite to the classification problems which have discrete results.

EXAMPLE: A typical example of this kind of problems could be predicting the amount of sales of a company in a period of time, given information of the previous months.

Clustering  Similarly to the classification case, the clustering consists in classifying the instances into categories. However, unlike the classification problems, it does not know beforehand to which category do the training instances belong, so it has to find the patterns inside the data itself. This makes it a typical unsupervised problem.

EXAMPLE: A typical example of this kind of problems could be trying to find different types of client behaviour from the company’s transactions.
2.2 Kernel Methods

2.2.1 Linear support vector machines

The support vector machine[11] (SVM) is a machine learning method solidly based on statistical learning theory. Intuitively, given a set of examples labeled into one of two classes, the linear SVM finds their optimal linear separation: this is the hyperplane that maximizes the minimum orthogonal distance to a point of either class (this distance is called margin of the separation).

In this project the notation $\langle a, b \rangle$ is used to express the inner product between two vectors $a, b$ (not necessarily finite-dimensional).

Consider a linearly separable i.i.d data sample $S = \{x_1, \ldots, x_N\}$ of training patterns (in $n$ dimensions), labelled into two classes $\omega_1, \omega_2$ by $z_1, \ldots, z_N$, with $z_i = +1$ if $x_i \in \omega_1$ and $z_i = -1$ if $x_i \in \omega_2$. If we set up an affine function $g(x) = \langle w, x \rangle + b$, then we have a linear discriminant as $\text{sgn}(g(x))$, for which we would like:

$$
\langle w, x_i \rangle + b > 0 \quad x_i \in \omega_1 (z_i = +1)
$$
$$
\langle w, x_i \rangle + b < 0 \quad x_i \in \omega_2 (z_i = -1)
$$

In short, $z_i(\langle w, x_i \rangle + b) > 0$, or $z_i g(x_i) > 0$, for all $1 \leq i \leq N$. Given the hyperplane $\pi : g(x) = 0$, the perpendicular distance from $x$ to $\pi$ is $d(x, \pi) = \frac{|g(x)|}{||w||}$. The support vectors are those $x$ closest to the hyperplane. Rescaling $w, b$ such that $|\langle w, x \rangle + b| = 1$ for these closest points, one obtains $|\langle w, x \rangle + b| \geq 1$. The support vectors
Theoretical Background

Implementation and study of a deep and wide Kernel Machine are now those \( \{ x_i / | \langle w, x_i \rangle + b | = 1 \} \).

The margin \( m(\pi) \) of a plane \( \pi \) can now be written as twice its distance of any support vector: \( m(\pi) = 2d(x_{SV}, \pi) = \frac{2}{\|w\|} \), where \( |g(x_{SV})| = 1 \). To maximize the margin, we should minimize \( \|w\| \) subject to \( z_i (\langle w, x_i \rangle + b) \geq 1 \), for all \( 1 \leq i \leq N \).

In the case where an hyperplane does not exist that can separate correctly the points in the data sample, a set of non-negative slack variables and a positive constant \( C \) are introduced to allow for small margin violations, leading to a soft margin:

\[
 z_i (w \cdot x_i + b) + \xi_i \geq 1 \quad i = 1, ..., N \tag{1}
\]

where \( \xi_i \geq 0 \). For an error to occur, the corresponding \( \xi_i \) must exceed unity, and so \( \sum_i \xi_i \) is an upper bound on the number of training errors. The optimal separating hyperplane can be found as the solution of the 1-norm Quadratic Programming problem:

\[
 \min_{w, \xi} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{N} \xi_i \\
 s.t. \quad z_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i, \quad i = 1, \ldots, N 
\]

The solution to this optimization problem corresponds to the saddle point of its associated Lagrangian:

\[
 \frac{\|w\|^2}{2} - \sum_{i=1}^{N} \alpha_i (z_i(\langle w, x_i \rangle + b) - 1 + \xi_i) + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \mu_i \xi_i \tag{2}
\]
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

where $\alpha_i, \mu_i \geq 0$ for $i = 1, \ldots, N$.

Once this QP problem is solved, the solution vector $w^*$ can be expressed as a linear expansion over the support vectors:

$$w^* = \sum_{i=1}^{N} \alpha_i^* z_i x_i$$  \hspace{1cm} (3)

The support vectors are precisely those $x_i \in S$ for which $\alpha_i^* > 0$ [11].

2.2.2 Nonlinear SVMs

Let us assume now that it is not possible to separate the training set with a hyperplane in input space, but that the SVM can project the inputs $x_i$ to a higher-dimensional space of characteristics $\phi$, using a nonlinear projection, wherein the training set is almost separable.

Here’s where the trick comes in: The input space data only appears in eqs. (2) and (3) in the form of inner products. 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 $\phi(x) \rightarrow \mathcal{H}$ be such a transformation (known as the feature map), where $X$ is the domain of the input data ($\mathbb{R}^d$ in our case) 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 $\phi(x)$ for $x$ in all positions in the solution. The problem of optimization of Equation (2) is now of the form

$$\max_{0 \leq \alpha_i \leq C} \left\{ -\frac{1}{2} \sum_{i,j=1}^{N} \alpha_i \alpha_j z_i z_j < \phi(x_i), \phi(x_j) > + \sum_{i=1}^{N} \alpha_i \right\}$$
Theoretical Background

Then the training algorithm, again, would only depend on the training set only through scalar products in $\mathcal{H}$, i.e., on functions of the form

$$< \phi(x_i), \phi(x_j)>$$

Now if there were a Kernel function such that $k(x_i, x_j) =< \phi(x_i), \phi(x_j)>$, we would only need to use $K$ in the training algorithm, and would never need to explicitly know $\phi$. This function is known as the kernel function.

Therefore, in the linear non-separable case, if one replaces $< x_i, x_j >$ by $k(x_i, x_j)$ everywhere in the training algorithm, the algorithm will produce a SVM which lives in $\mathcal{H}$. Finally a classifier in $\mathcal{H}$ is

$$f(x) = sgn(<w^*, x > + b^*) = sgn\left( \sum_{i=1}^{n} \alpha_i^* y_i k(x_i, x) + b^* \right).$$  \hspace{1cm} (4)

The great advantages exhibited by SVMs can be summarized as:

- SVMs are based on a solid theory and on the principle of minimization of the structural risk.

- SVMs do not have local optima and have few parameters to fit, usually $C$ and the Kernel.

- The SVMs are resistant to overfitting 5.3 (although not immune); overfitting is more probable when the parameter $C$ is larger than necessary.

- The input vectors need not be real numbers; they can be anything as long as one can devise a kernel.
2.2.3 Existent Kernels

Amongst the most widely used and well-known Kernels we find:

- Linear Kernel
  \[ k(u, v) = \langle u, v \rangle \]  
  \[ (5) \]

- Polynomial Kernel
  \[ k(u, v) = (\langle u, v \rangle + \gamma)^d \]  
  \[ (6) \]
  with \( \gamma \geq 0 \in \mathbb{R} \) and \( d \in \mathbb{N}^+ \) parameters.

- Gaussian Kernel, also known as Radial Basis Function (RBF) kernel
  \[ k(u, v) = e^{-\frac{||u-v||^2}{\sigma^2}} \]  
  \[ (7) \]
  with \( \sigma \in \mathbb{R} \) parameter.

- Sigmoidal Kernel
  \[ k(u, v) = \tanh(\alpha \langle u, v \rangle + r) \]  
  \[ (8) \]
  for some (not every) \( \alpha > 0 \) and \( r < 0 \) parameters, where \( \tanh \) is the hyperbolic tangent function.

The RBF is by far the most popular choice of kernel in Support Vector Machines. This is mainly because of its localized and finite response across the entire range of the real line; it also includes the polynomial Kernel as a limiting case. All these Kernels assume and need the data set features to be continuous.
2.3 Deep Learning

Deep Learning is a branch of Machine Learning specialized in solving high complexity problems. The algorithms needed in order to do so require high levels of abstraction, in order to model the complex patterns in these problems. This abstraction is achieved by composing multiple layers of more traditional transformations.

The underlying concept in Deep Learning is an extension of the principle of distributed representations, which states that observed data is generated by the interactions of many different factors on different levels. Deep Learning assumes also that these factors are grouped into different levels, each of them more abstract than the previous one. This is the reason behind Deep Learning multiple layers, as it is trying to reproduce this abstraction.

This is achieved by first applying a transformation to the input, and then providing the output as the input to the next layer. Each of this steps apply a non-linear transformation to the input it receives, allowing for more expressive feature spaces.

Neural Networks is one of the most commonly used Machine Learning algorithms, in which the main element are nodes, called neurons, which are grouped into layers. The input is provided to the first one, and the final layer returns the output, which can be used directly or applying some processing in it. At first sight, it may look as a Deep Learning algorithm, but it has been proven impossible to train more than a small number of layers and produce good results. Since they ap-
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

peared, however, there has been interest in training multiple layers, so there have been many different approaches, and in recent years, Deep Learning has become a reality. Because of this, many Deep Learning methods use neural networks in their layers, despite some different methods having been used.

2.3.1 Kinds of Deep Learning

2.3.1.1 Supervised Learning

When used in a supervised manner, Deep Learning differs greatly from the most common algorithms, as the dataset treatment explained in the section 2.1.1.2 of this document, in which the parameters, or features, had to be almost hand-picked, becomes unnecessary. This process is rather labor-consuming, varies on every problem attacked and may require many tries before finding the right choice. Instead, Deep Learning works on the raw data, so it does no require any costly pre-treatment, producing intermediate representation which lead to the final result. This, together with the high level of abstraction reached, serves as an example of the potential of Deep Learning in many areas, such as image recognition.

2.3.1.2 Unsupervised Learning

On the other hand, Deep Learning can also be used in an unsupervised manner, that is, with unlabeled data. As unlabeled data is usually more common than labelled one, it is of great importance to have powerful
Theoretical Background

algorithms able to work with it. In this sense, Deep Learning allows finding complex patterns in unlabelled data, which, together with the unnecessity of hand-picking its features, reduces in some way the need of human tuning in the training process.

2.3.2 Most common Deep Learning algorithms

Below there is a list of the most commonly used Deep Learning methods:

- Deep neural networks [14]: They are "simple" neural networks combined forming multiple layers. They are typically trained in a feedforward way, that is, each layer simply feeds their outputs to the next one. Though it is the most "simple" Deep Learning algorithm, it has two main drawbacks, which are computation time, and a tendency to overfitting. However, there are ways of minimizing these problems, so they do have applications.

![Deep Neural Network](image)

Figure 2: Deep Neural Network [5]

Figure 2 illustrates a typical Deep Neural Network scheme: notice how all transitions are done in a single direction.
Theoretical Background

Deep belief networks [17]: At first sight they look similar to Deep neural networks, as they are also composed of many layers of artificial neural networks. However, the main difference is that Deep belief networks is not trained in just a feedforward way, and connections between layers work in both ways. Particularly, each layer is trained individually in an unsupervised way, and afterwards backpropagation is used in order to fine tune the weights. The usual neural network used is called Restricted Boltzmann Machine (RBM [15]).

Figure 3: Typical Deep RBM scheme [2]

Figure 3 illustrates a typical Deep Learning scheme: a greedy unsupervised layer-wise pre-training stage followed by a supervised fine-tuning stage affecting all layers.

- Auto-encoders: The idea behind auto-encoders consists in trying to find a good representation. Basically, the encoder is a function $f$ such that maps the input $x$ to the output $y$, while a decoder performs the inverse, mapping from output $y$ to input $z$. The
aim of the auto-encoder is to find a representation such that the error between $\mathbf{x}$ and $\mathbf{z}$ is minimized.

![Auto-encoder scheme](image)

Figure 4: Auto-encoder scheme [6]

Figure 4 shows how does an Auto-encoder work. Notice how there is as much expressiveness in the few nodes in the bottleneck as in the whole input nodes.

There are many other types of Deep Learning methods, but these are some of the most representative ones.

### 2.4 Deep Kernel Methods

It has already been explained the way Deep Learning has used neural networks in their layers. However, training of these methods require difficult nonlinear optimizations and multiple heuristics. Kernel methods, on the other hand, are much easier to train, requiring only solving comparatively simple quadratic problems, by means of the *kernel trick*. This is a common technique in machine learning, which allows us to implicitly perform transformations into a higher (even infinite) dimensional spaces, while still only working with the vectors in the input space. This technique is used to augment and improve the modeling
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

capabilities of a variety of Machine Learning algorithms, such as Support Vector Machines [11]. By transforming the data and adding more features, it might be possible to project the data into a higher dimensional space where the data is indeed linearly separable. This is why at first sight it resulted interesting to try and use them in Deep Learning, which would seemingly make the most of both the expressivity of kernels, and the capacity of Deep Learning of learning higher level features.

There have been recent advances in this idea, one of the most important being the one involving a whole new kernel family, called arc-cosine kernel. This kernel seems to benefit from Deep Learning, while being much easier to train than neural networks. In the following section, this kernel is explained more thoroughly.

2.4.1 Arc-cosine Kernel

The arc-cosine kernel [1], as many kernels do, computes the similarity of two vectors, $x, y \in \mathbb{R}^d$. It requires a parameter, called $n$, and the formula of the $n$th kernel is:

$$k_n(x, y) = 2 \int dw e^{-\frac{||w||^2}{2}} \frac{1}{(2\pi)^{d/2}} \Theta(<w, x>)\Theta(<w, y>)(<w, x>)^n(<w, y>)^n$$

being $\Theta(z) = \frac{1}{2}(1 + \text{sign}(z))$. This kernel, the basic properties of which will be explained in the following sections, has interesting connections with neural networks, which makes it a good candidate to
benefit from Deep Learning.

### 2.4.1.1 Basic properties

The resolution of the previous integral results in the following formula:

$$k_n(x, y) = \frac{1}{\pi} ||x||^n ||y||^n J_n(\theta)$$  \hspace{1cm} (10)

which shows a trivial relation between $x$ and $y$ magnitudes, but a more complex, and interesting, relation via the angle ($\theta$) between them:

$$\theta = \cos^{-1}\left(\frac{\langle x, y \rangle}{||x|| \cdot ||y||}\right)$$  \hspace{1cm} (11)

The function $J$ is expressed as follows:

$$J_n(\theta) = (-1)^n (\sin \theta)^{2n+1} \left( \frac{1}{\sin \theta} \frac{\partial}{\partial \theta} \right)^n \left( \frac{\pi - \theta}{\sin \theta} \right)$$  \hspace{1cm} (12)

where $(-1)^n \left( \frac{1}{\sin \theta} \frac{\partial}{\partial \theta} \right)^n \left( \frac{\pi - \theta}{\sin \theta} \right)$ is the $n$th partial derivative of $\left( \frac{\pi - \theta}{\sin \theta} \right)$ with respect to $\cos \theta$.

An interesting property, used when the code was optimized, is the following:

- $k_0(x, x) = 1$
- $k_1(x, x) = ||x||^2$
- $k_n(x, x) \propto ||x||^{2n}$
2.4.1.2 Similarities with neural networks

In this section there is an intuition of the similarities between arc-cosine kernel and neural networks, which makes them so appealing.

In order to see this relation, some explanation about neural network is provided. A typical single layer neural network looks can be seen in Fig. 5:

![Figure 5: Single-layer Neural Network](image)

in which the input $\mathbf{x}$ is mapped into the output $f$ (composed of $m$ nodes, which are usually called neurons) by means of a non-linear pairwise transformation. This transformation uses the matrix of weights $W$, whose position $W_{ij}$ connects the $i$th input to the $j$th output, so that, in fact, $W$ is a $m \times d$ matrix.
The output of the single layer, which is usually called hidden layer, is, as seen, $f$, which is expressed as:

$$f(x) = g(Wx)$$ \hspace{1cm} (13)

, being $g$ a non linear activation function. In this particular case, the family of activation functions used is $g_n(z) = \Theta(z)z^n$, which can be seen in Figure 6.

![Figure 6: Activation functions](image)

Now focus on how inner products are transformed by the mapping in single-layer threshold networks. Being $w_i$ the $i$th row of the weight matrix $W$, that is, the vector in charge of weighting the input vector $x$ to the $i$th neuron $f_i$, $f(x)$ can be expressed as:

$$f_n(x) = \sum_{i=1}^{m} \Theta(<w_i,x>)(<w_i,x>)^n$$ \hspace{1cm} (14)

, where $m$=number of output units. Therefore, the inner product
between different outputs of the network results in:

\[ < f(x), f(y) > = \sum_{i=1}^{m} \Theta(< w_i, x >)\Theta(< w_i, y >)(< w_i, x >)^{\alpha}(< w_i, y >)^{\alpha} \]

(15)

Supposing a neural network with an infinite number of neurones, and assuming \( W_{ij} \) is distributed as a standardized multi-variate Gaussian distribution, it can be seen as (15) is equivalent to (9) using:

\[ \lim_{m \to \infty} \frac{2}{m} < f_n(x), f_n(y) > = k_n(x, y) \]

(16)

At this point, it can be said that the arc-cosine kernel function behaves as the inner product between feature vectors derived from the mapping of an infinite single-layer threshold network.

2.4.1.3 Multiple layer arc-cosine kernel

In this section there is an explanation of the intuition behind the multi layer aspect of arc-cosine kernels, as well as the formula obtained. A kernel can be viewed as a non-linear mapping from \( x \) to another feature vector \( \phi(x) \). This is achieved via the kernel trick, which allows for easily compute \( k(x, y) = < \phi(x), \phi(y) > \) without explicitly knowing \( \phi \). Therefore, in order to obtain the equivalent of multi-layer kernels, this formula must be composite with itself. The motivation for this is that, if a "simple" arc-cosine kernel simulates the behaviour of a single layer infinite neural network, a composition of \( l \) mappings of the same kernel will mimic the behaviour of a multi-layer neural network. Basically, the
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

Goal is to obtain, for the arc-cosine kernel:

\[ k^{(l)}(x, y) = \langle \underbrace{\phi(\phi(\cdots \phi(x)))}, \underbrace{\phi(\phi(\cdots \phi(y)))} \rangle > \quad (17) \]

being \( l \) the number of layers simulated, that is, \( l \) consecutive non-linear mappings.

This composition can be applied to kernels other than the arc-cosine kernel, which will form the basis of our own contribution in this work. First, let us analyse what happens when we compose other, more standard kernels with themselves:

- **Linear kernels:** This kernel, expressed by the formula \( k(x, y) = \langle x, y \rangle \) is trivial to compose. Basically, composing \( \phi(\phi(x)) \) results in: \( \phi(\phi(x)) = \phi(x) = x \), that is, the identity mapping. Therefore, it does not produce any interesting result.

- **Polynomial kernels:** This kernel, taking the form of \( k(x, y) = (\langle x, y \rangle)^d \), when composite results in:

  \[ \langle \phi(\phi(x)), \phi(\phi(y)) \rangle = (\langle \phi(x), \phi(y) \rangle)^d = (\langle x, y \rangle)^{2d} \]

This result does not yield any particular interest, as it is only a higher degree polynomial than the one of which it was composite.

- **RBF kernels:** This kernel, with the form of: \( k(x, y) = e^{-\lambda ||x-y||^2} \), may be more interesting than the two above, and its composition results in:

  \[ \langle \phi(\phi(x)), \phi(\phi(y)) \rangle > = e^{-\lambda ||\phi(x) - \phi(y)||^2} = e^{-2\lambda(1-k(x,y))} \]
Theoretical Background

Admittedly, this result can not be trivially analyzed, but some analysis makes clear that it does not produce any interesting effect [1].

As it can be seen, even though kernel composition can be applied to many different kernels, it does not necessarily improves its expressiveness.

Once most of the most well known kernels have been discarded, it is the turn of the arc-cosine kernel. This composition is not so trivial as the previous ones. First, there is the case with \( l = 1 \), given by the equation (10). The case \( l + 1 \) is given by [1]:

\[
k_{l+1}^n(x, y) = \frac{1}{n} \left[ k_n^{(l)}(x, x)k_n^{(l)}(y, y) \right]^{n/2} J_n \left( \vartheta_n^{(l)} \right)
\]  

(18)

, being \( \vartheta_n^{(l)} \) the angle between \( x \) and \( y \) in the feature space in the \( l \)th "layer" of the kernel. It is given by the formula:

\[
\vartheta_n^{(l)} = \cos^{-1} \left( k_n^{(l)}(x, y) \left[ k_n^{(l)}(x, x)k_n^{(l)}(y, y) \right]^{-1/2} \right)
\]  

(19)

For simplicity reasons, in this formula there is the assumption that the degree \( n \) is the same in each composition, but if desired, it would not be difficult to add the possibility of specifying it for each layer. However, as it was not expected to add much benefit, it was obviated, leaving its testing to further work.

When trying this composition, it produced interesting results, as it mimicked the behaviour of multi-layer neural networks, being extremely easier to compute, and quicker to train.
2.5 Extensions

This project also aimed to introduce some modification in the original article, trying to expand the method proposed. In this sections there are described the expansions studied, including both the intuition, theoretical background and the implementation.

2.5.1 RBF and Arc-cosine kernels composition

In this section, the kernel that is being composite with be expressed as: $k(x, y) = \langle \phi(x), \phi(y) \rangle$, whereas the kernel with which it is being composed is expressed as: $\tilde{k}(x, u) = \langle \tilde{\phi}(x), \tilde{\phi}(y) \rangle$

As explained in section 2.4.1.3, arc-cosine kernel mimics the behaviour of multi-layer neural networks by being composite with itself. What has been tried in this project is to try different kinds of compositions, and see how they behave. Each composition consisted in substituting the $x$ and the $y$ of the first kernel with the $\phi(x)$ and $\phi(y)$ of the second one. The motivation behind this is, in part, the interest in seeing how these composition behave but, mainly, the fact that the arc-cosine kernel does not have any adjustable parameter aside from the number of layers and assuming the degree is fixed. This feature, which tends to be common in the family of the infinite kernels, means that it is fast to find their optimal configuration but, at the same time. In case it does not behave properly in a problem, it is difficult to make any improvement. However, if the arc-cosine kernel were to be composite with another kernel which did have a parameter, this particular
Theoretical Background

Implementation and study of a deep and wide Kernel Machine problem would be overcome. Concretely, three compositions have been tried:

1. **Arc-cosine composite with RBF**: Using as a reference the equation (10) the basic form of a composite arc-cosine kernel is:

   \[
   k_n(x, y) = \frac{1}{\pi} ||\phi(x)||^n ||\phi(y)||^n J_n(cos^{-1}\left(\frac{\langle \phi(x) \cdot \phi(y) \rangle}{||\phi(x)|| \cdot ||\phi(y)||}\right))
   \]

   \(20\)

   In this base case, it is not immediate to calculate and, sometimes, it can be even impossible, so some transformation had to be introduced in order for it to be easily obtained. The first transformation consisted in getting rid of the \(||\phi(x)||^n \) and \(||\phi(y)||^n\.

   The process followed consisted in doing:

   \[
   ||\phi(x)||^n = (||\phi(x)||^2)^{n/2} = (\langle \phi(x), \phi(x) \rangle)^{n/2} = (\tilde{k}(x, x))^{n/2}
   \]

   \(21\)

   , being \(\tilde{k}\) the kernel with which arc-cosine kernel is composite. This way, instead of having to compute the coordinates of \(x\) in the transformed feature space determined by \(\phi\), now there is only the value of a kernel.

   Finally, using this same logic, the \(\phi\)s determining the angle were
substituted in the following way:

$$< \phi(x) \cdot \phi(y) > = \frac{\tilde{k}(x, y)}{\sqrt{\tilde{k}(x, x) \cdot \sqrt{\tilde{k}(y, y)}}}$$

(22)

After all this transformations, the resulting formula is feasible to calculate:

$$k_n(x, y) = (\tilde{k}(x, x))^{n/2}(\tilde{k}(y, y))^{n/2}J_n \left[ \cos^{-1} \left( \frac{\tilde{k}(x, y)}{\sqrt{\tilde{k}(x, x) \cdot \sqrt{\tilde{k}(y, y)}}} \right) \right]$$

(23)

The last step was substituting $\tilde{k}(x, y)$ by the RBF kernel, whose equation is (7). It is important to point out the fact that the RBF kernel is normalized, which means, amongst other properties, that it fulfills $k_{RBF}(x, x) = 1$. This property is interesting in this case, as it simplifies considerably the formula, which becomes:

$$k(x, y) = \frac{1}{\pi} \cdot 1 \cdot 1 \cdot J_n(\cos^{-1} \left( \frac{\exp(-\lambda \|x - y\|^2)}{\sqrt{1 \cdot 1}} \right)) = \frac{1}{\pi}J_n \left( \cos^{-1}(\exp(-\lambda \|x - y\|^2)) \right)$$

(24)

2. **RBF kernel composite with arc-cosine**: In this case, the process consisted in the opposite of the previous case, as it was the RBF the formula the $x$ and $y$ of which were replaced by $\phi(x)$ and $\phi(y)$. First of all, the initial composite formula for RBF is the following:

$$k(x, y) = \exp(-\lambda \|\phi(x) - \phi(y)\|^2)$$

(25)
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

In this situation, also, some transformations are needed, as $\phi(x) - \phi(y)$ is unknown. The process followed in order to obtain a computable substitution can be found below:

$$||\phi(x) - \phi(y)||^2 = <\phi(x) - \phi(y), \phi(x) - \phi(y)> =$$

$$||\phi(x)||^2 + ||\phi(y)||^2 - 2 <\phi(x), \phi(y)> =$$

$$\tilde{k}(x, x) + \tilde{k}(y, y) - 2\tilde{k}(x, y)$$

In order to understand these steps:

- The inner product between two vectors $x, y \in \mathbb{R}^d$

$$<x, y> = \sum_{i=1}^{n} x_i y_i$$

(27)

- The Euclidean norm of a vector $x \in \mathbb{R}^n$, expressed as $||x||$, is:

$$||x|| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

(28)

- Therefore, it can be seen that:

$$||x||^2 = <x, x>$$

(29)

which lead to the formula for the RBF kernel composite with another kernel $\tilde{k}$:

$$k(x, y) = \exp(-\lambda(\tilde{k}(x, x) + \tilde{k}(y, y) - 2\tilde{k}(x, y)))$$

(30)

The only remaining step is to substitute $\tilde{k}$ by the arc-cosine kernel, but, as no interesting simplification can be achieved, the whole substitution in not provided here.
3. **RBF kernel composite with normalized arc-cosine**: As it has already been pointed out, RBF is a normalized kernel, which provides some interesting properties. Therefore, the logical step was to try and normalize the arc-cosine kernel. First of all, the normalized formula for a kernel is:

\[
\tilde{k}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x)}\sqrt{k(y, y)}}
\]  

(31)

In the case of the arc-cosine kernel, it is important to remember that the arc-cosine kernel fulfills \( k(x, x) \propto ||x||^{2n} \). Therefore, if (31) is developed with the arc-cosine kernel, evolves into:

\[
\tilde{k}_n(x, y) = \frac{1}{\pi} \frac{||x||^n ||y||^n J_n(\theta)}{\sqrt{||x||^{2n}} \sqrt{||y||^{2n}}} = \frac{1}{\pi} \frac{||x||^n ||y||^n J_n(\theta)}{||x||^n ||y||^n} = \frac{J_n(\theta)}{\pi}
\]  

(32)

Which, as it can be seen, results in a simpler formula than the non-normalized one. If added to the RBF composite formula, the formula obtained becomes:

\[
k(x, y) = \exp\left(-\lambda(1 + 1 - 2\frac{J_n(\theta)}{\pi})\right) = \exp\left(-2\lambda\left(1 - \frac{J_n(\theta)}{\pi}\right)\right)
\]  

(33)
Theoretical Background

Implementation and study of a deep and wide Kernel Machine

It is interesting to notice that, as the $\lambda$ parameter is adjustable, it absorbs the constant 2 and, therefore, the formula can be simplified into:

$$
exp\left( -\lambda \left(1 - \frac{J_n(\theta)}{\pi}\right) \right) = exp\left(\lambda \left(\frac{J_n(\theta)}{\pi} - 1\right)\right) \tag{34}
$$

As it can be seen, in all three of this compositions there is an additional parameter, called $\lambda$, which allows for some adjustment in the kernel.

Unfortunately, due to time constraints it proved impossible to try these composites, so there is no result to evaluate their performance. However, in case the project was continued in the future, this is one of the first things to be tried. The methods required to try them are already implemented, but it lacked the necessary execution time.
3 Project Management

3.1 Obstacles and risks

In this project there were two main possible obstacles that could negatively affect to its development or usefulness:

- The first one would be that the results obtained ended up being useless, in case they were much worse than the other methods with which they were compared. This would not be exactly an obstacle, but it would make the project almost lacking of utility. However, as this was impossible to predict, and the results shown in the consulted article seemed promising, the risk was accepted. Moreover, even if the results were extremely bad, at least it would have allowed the student to familiarize with the area of Deep Learning, meaning that it would still have been useful.

- The second possible problem in this project would appear in case the implementation of the method were slower than it had been planned. Were this to happen, there would be fewer time to analyze it. In order to minimize the impact of this, in case the work rhythm appears lower than expected, the workload should be increased and, in case even this proved not enough, some of the extensions or optimizations planned would be discarded, focusing instead in obtaining the functional method explained in the article. Additionally, even though it would be interesting to
try it in more than one Machine Learning problem, it may not be possible, so in case there is not enough time, the student will focus in only one problem

3.2 Following of the project

As it has already been mentioned, the project has been supervised by the project director, the doctor Lluís Belanche [8], from the UPC [9]. In order for him to be able to properly supervise the project, it was important to hold frequent meetings, so that it can guide and advise the student. Additionally, in case the project started deviating from its original purpose, these meetings would allow to detect and correct it or, in case this deviation were actually necessary, to ratify the new direction of the project.

On the other hand, in the absence of such meetings, e-mails were the mail communication tool, allowing for quick queries of petitions. However, e-mails by themselves could not completely substitute the meetings, being, instead, a complement for them.

3.3 Schedule

The project needed an initial schedule, in order to be able to make a better control of its progress. The whole project took 4 months, from February 2015 to June of the same year. In this kind of project, it was difficult to elaborate a complete and precise schedule of every task, but some guidelines were necessary, in order to have a reference. This way
it was possible to detect if the progress was done according to the initial idea, or if the work rhythm must be increased. The initial guideline was divided into a certain number of tasks, which are listed below. They are listed in chronological order, but some of them were done in parallel:

- Project Management: As a part of the bachelor degree final project, there was a non-attended subject that the student had to do, called Project Management. This subject taught the student how to completely plan a project, from the timetable to the economical management. It is thought to be done before the project starts, so it was the first task to be done. During its duration, there were some deliverables to be done (7 of them), every one of which covered an essential aspect in the planning of a project. The duration of every deliverable varied from one to another, but it was expected to take around 75 hours for the student to do them, distributed as follows:

  - Range of the project: 9.25h
  - Schedule of the project: 8.25h
  - Economical management and sustainability: 9.25h
  - Video presentation: 6.25h
  - Contextualization and bibliography: 15.25h
  - Project justification: 8.5h
  - Oral presentation and final deliverable: 18.25h
- **Documentation research**: At the same time than the Project Management subject, the documentation required for the project had to be studied, as some theoretical background was needed. This documentation ranged from academical reports, including the one this project is based on [1] to other possible sources such as Wikipedia. This stage did not have a fixed duration, as it was likely that, during the development of the project, more documentation was required. Thus, this task was not exactly a requirement for the next task in its whole, but, instead, a part of the documentation had have to be read before continuing, and the rest was be done during all the project.

Due to this factors, it was difficult to calculate the amount of time required for it, but it was assumed that somewhere around 30 hours for the first part was a good start, and around 40 additional hours would be probably necessary before the project was finished.

- **Implementation of a functional model**: Once the first part of the documentation was studied, the first step was to obtain a functional model for the method to develop. This model was expected to have a reasonable execution time, and be applicable to some Machine Learning problem, performing well enough on them. However, being a first model, it was not expected to perform extremely well and, instead, to provide a base from which to develop further optimizations. This optimizations would affect
both the performance when applied to problems and the execution time of the program, and would be done in the following task.

The implementation of this method was expected to take around 120 hours, but as most of the implementations, it was difficult to anticipate its duration, as many unforeseen circumstances may arise.

- **Optimization of the code** Once the working model was obtained, the next step, as already explained, was to optimize it. There are two main areas to improve:
  
  - **Execution time**: Machine Learning methods in general, and Deep Learning in particular, usually have high computational requirements, which, in some cases, may be around the days of execution. The method to develop was not expected to take so much time, but, anyway, it was likely to require a certain amount of time. Therefore, it was important to make it as fast as possible, to be able to execute it as many times as it was necessary to analyze it.
  
  - **Performance**: Even more important than the execution time, the performance of the model must be improved as much as possible. This performance was measured in the precision displayed when analyzing data with the model obtained, which is the whole purpose of Machine Learning, to obtain predictions and results from data.
This stage was likely to be done in parallel with the following one, that is, the analysis of the models, as the improvements were to be done according to its behaviour. Additionally, some documentation was probable to be required, meaning that at this point, it was likely to have three tasks being done simultaneously.

The expected required time for this task was of 100 hours, but it depended on the opinions of both the director of the project and the students, as they would judge the amount of time assigned depending on the circumstances. In case the project was much behind the schedule, this stage might be reduced, but the intention was to improve the code as much as possible.

- Analysis: This task, corresponding to the last stage of the program itself, consisted in the analysis of the results obtained by the program, comparing them with the ones obtained by other existing methods in order to check their quality. The analysis took into account factors such as precision, execution time, and number of data required for it to work (as quality data is difficult to obtain, the less it is required the better). However, the analysis were not to be exhaustive, but to provide an intuition of how to continue the optimizations. Once the final program was obtained, however, the analysis would be significantly more exhaustive than the others.

The whole process of analysis, design of the experiments, and improvements and conclusions design was expected to consume
as much as 100 hours of work.

- **Final stage:** In the last stage of the project, all the work done until then had to be put together, prepare the final inform, and to prepare the oral defense of the project in front of the tribunal. In order to start it, all previous stages had to be obviously done. As this stage consisted basically in a gathering of results, it was expected to take no much more than 30 hours.

### 3.3.1 Dependency between tasks

As it has already been mentioned, some tasks required others to be finished in order for them to begin. In the following table, this dependencies are specified, even though, as explained, some of them will be done in parallel.

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Management</td>
<td>-</td>
</tr>
<tr>
<td>Documentation research</td>
<td>-</td>
</tr>
<tr>
<td>Implementation of a functional model</td>
<td>Documentation research (though only partially)</td>
</tr>
<tr>
<td>Optimization of the code</td>
<td>Implementation of a functional model</td>
</tr>
<tr>
<td>Analysis</td>
<td>While trying the program: Implementation of a functional model  For the final program: Optimization of the code</td>
</tr>
<tr>
<td>Final stage</td>
<td>Analysis</td>
</tr>
</tbody>
</table>
3.3.2 Deviations and Corrections

In a project such as this, it was reasonable to assume that the schedule could not be completely followed, and some deviations from it would appear. This deviations could be either positive ones, such as a task requiring less time than the assigned, or negative, in the opposite case. The first case would not be any problem, but the contrary, as if a task were finished before the scheduled, the logical consequence is to continue with the following task. However, should a task require more time than the expected, it would produce a delay in the project and, in case this delay was extremely large, it could make a whole task impossible to fulfill.

In order to minimize the negative impact of this situation, it was fixed as a priority the fulfillment of the tasks Implementation of a functional model and Analysis. Therefore, the Optimization of the code task depended in great measure on the other two tasks to be finished in their expected time.

Additionally, the analysis would be done in an incremental way: starting with a single problem, which would be use to try the behaviour of the problem, and when the program was finished, it would be tried with that same problem and, in case it was still possible, with different ones. This way, even though there was little time to try the program, at least a whole problem could be analyzed.

This deviations could be actually seen as a kind of redistribution of time, as they reduce the number of hours spent in a task while
increasing the time working in another task. However, the intention was to completely fulfill all tasks, meaning that, to a certain extent, an increase of the total of hours spent was accepted beforehand. This extra time could mean that, even though a task needed more time than the assigned at first, all tasks could be completed.

3.3.3 Changes on the schedule

During the development of the project, some minor changes in the initial schedule had to be done due to the faced circumstances. Basically, in the initial planning the implementation was divided into two stages. As already explained, the first stage would focus on obtaining a basic but functional program, which would be partially analyzed, while the second one would focus on improving and optimizing this first model. However, when the first stage was already finished, the schedule had to be slightly modified. The reason for this change was that the initial model, despite obtaining good results, was extremely slow, so it was impossible to perform a minimum analysis in a reasonable amount of time. Therefore, the analysis of this first model was obviated, apart from checking its precision to see that it actually worked. Instead, as soon as its slowness proved an obstacle, the next stage was already started, focusing only in increasing as much as possible the speed of the code.
3.3.3.1 Consequences and justification

The improvements introduced in the code due to this circumstances allowed for a more exhaustive analysis than any that could have been done with the previous code, and in a shorter span of time. Therefore, the change in the initial schedule was completely justified. Anyway, the initial schedule was not excessively modified, as the order of the tasks has been more or less maintained. The main difference lays in the fact that the improvements in the code have been done before analyzing the first code obtained, and they became an essential part of the project instead of a simple optional improvement.

The consequences for the project, if any, were essentially positive, as it actually allowed for some time saving, as it made the analysis of the first code unnecessary.

3.4 Sustainability

In order to evaluate the sustainability of the project, it has been divided into 3 aspects that could be said to represent it: the economic, social and environmental aspects. Each of these aspects has been individually evaluated.
3.4.1 Economical aspect

This project is considered to be very sustainable in the economical area, because of the following reasons:

- The cost of the project, without taking into account any possible salary, is almost 0, as the specific equipment required is close to null. Therefore, in this sense, this project is close to optimal.

- Should the cost be reduced, the fact that the project is divided in different tasks allows for an easier choice of which to modify or reduce.

- Once finished, the upkeep cost of the project would be almost zero:

  - In case an update was necessary, it could require the payment of the engineer who did it. However, these updates are expected to be very few, if any.

  - For every different problem in which to apply this method, it would require a whole training of the model, meaning some expert knowledge would be necessary. This knowledge would include both the engineer who trained it, and the expert in the problem being analyzed. This would suppose additional costs, but as every potential user would probably need few, if not only one, problems, the cost for every user would be small.
- In case good results were obtained, it could hypothetically, improve the quality of predictions in many different areas, such as medicine, image recognition or so. Even though it is an unlikely situation, in case it happened, the inversion would surely be recouped.

3.4.2 Social aspect

In the social aspect this project is not expected to do a great contribution, but its sustainability is reasonable. The reasons are the following:

- The social and politic situation of the country is stable, even though the economy is impaired due to the economical crisis.

- Machine Learning in general is becoming more and more popular as time passes.

- This project does not aim to cover any concrete necessity, even though it does have the potential to produce long term benefits.

- The R+D sector, to which this project belongs, is of the utmost importance to generate income in a country. However, the areas in which a project of this features could potentially influence are much wider, ranging from the medicine to the financial world, but including any area in need of an analysis of complex data.

- In the hypothetical case this project improved the state-of-art results of a certain problem, and it were applied to real cases, it could in the end benefit final user.
- Due to Machine Learning features, an area with many sub-areas and multiple alternatives, it is almost impossible that this project produces an important change. However, it would not produce any disadvantage neither, and the results of this project could inspire further research.

- Under no circumstances would this project produce harm to any individual nor collective.

3.4.3 Environmental aspect

This project is highly sustainable in the environmental aspect, due to the following reasons:

- The impact to the environmental consists mainly in the electricity used by the computer in which the project has been developed. Anyways, as this computer is a laptop, this consumption is not expected to be high.

- No waste is produced apart from any possible note that could be taken during the project (paper that, anyhow, would be recycled).

- The only additional equipment acquired is a notebook, nor hardware nor office supplies.

- As the result is a software product, it will not affect in any way the environment, and in case it has to be disposed of in a future, it will be easily done.
- Even though it will not actively benefit the environment, it will not harm it in any way.

- Finally, this project is not a target in itself, but it is thought to be used later in other fields in order to obtain predictions. Therefore, it is expected to be used again.
4 Project Development

In this section there is a description of how did the project develop. As it has been already explained, one of the main goals was trying to replicate the results obtained in the original article. Therefore, the first step was to create a program implementing the method explained, but at the same time reasonably efficient, so that execution times remained acceptable.

4.1 Programming Language Used

One of the first decisions taken was the programming language used, which was C++. The main reason for this choice was its speed, being as it is one of the most, if not the most, fastest languages. Additionally, it is widely used, which made it easier to work with.

4.2 SVM-Solver

As explained in the section 2.4.1 of this document, the contribution of Cho and Saul is the study and development of the arc-cosine kernel to simulate multiple neural network layers. However, this kernel needs to be used in a Support Vector Machine in order to work as a Machine Learning method. Programmes solving Support Vector Machine problems are widely available, which allowed focus on developing the kernel itself, instead of having to implement the whole program, which was not the goal of this project.
4.2.1 *libsvm*

From all the available Support Vector Machine solvers, the one chosen was *libsvm* [20], a library compatible with many different programming languages, including C++. This open source library allows using some of the most known kernels, including RBF or the sigmoid ones, and it also provides the option of using your own defined kernel.

This last option was, obviously, the one that it was being sought, but the way it had to be used was not the expected one, as it had to receive the **kernel matrix** in order to work. This matrix $M$ consists in a $N \times N$ matrix, where $N$ is the number of instances provided, and $M_{i,j} = k(x_i, x_j)$. In a dataset with $N=60000$ such as MNIST, the one used in this project, that would make $60000 \times 60000 = 3.6 \times 10^9$ positions, which was impractical in terms of memory usage.

4.2.1.1 Adaptation of *libsvm*

After analyzing the work done by Cho & Saul, the solution found was modifying the code of the *libsvm* library so that it provided the option of using the arc-cosine kernel. This way it avoided having to provide the kernel matrix, saving both time and memory, and making it easier to use in the future.

In the aspect of the copyright, *libsvm* is provided as open source, as long as the given copyright is maintained in the resulting code, so there would be no problem for doing this modification in it.

The way it was implemented, the code of the arc-cosine kernel is in
a separate class, so that it could be easily modified and remained as modular and independent as possible.

4.3 Arc-cosine Kernel

As it has been mentioned, the arc-cosine kernel has been implemented, in which there is all the necessary methods in order to obtain its value. However, there have been two versions of this class, the one made initially, and the one that optimized it and that was finally used.

4.3.1 Initial Implementation

Initially, the arc-cosine kernel class consisted of many functions, every one of which computed a certain part of the kernel in its exact way, as it was declared in the article. This way, the code was easily read, it was extremely quick to identify which part was failing in case something did not work, and, at first, it looked like it would be easy to optimize, modifying a function at a time. At this stage, the program was tried thoroughly, creating models and predicting new instances with them, until a functional version was obtained. The criteria to decide if it was acceptable in terms of precision was not very clear, although an essential requirement was performing better than the random choice. However, in the end no longer consideration was needed in this aspect, as once the arc-cosine kernel was correctly implemented, the accuracy of the obtained model jumped to practically its peak. Therefore, it was easy to determine when to start optimizing the code.
4.3.2 Final Implementation

Despite the good results obtained, in the end the implementation obtained proved extremely inefficient in terms of computational time. That is why optimizing the code became not an option but a must. Even though at first this optimization was attempted analyzing functions individually, not much progress was done in this way. Thus, in the end it was decided to modify completely the code, suppressing completely the use of functions (which added a non negligible overhead) and making the most of the arc-cosine features to reduce the number of required operations. Some of the optimizations are listed below:

- Completely suppressing the use of functions

- Replacing all calls to $k(x, x)$ by $<x, x>$, cheaper to compute

- Implementing each degree individually, which allowed for some simplification of the formulas.

- Substituting costly operations (such as $\text{pow}(x, 2)$) by cheaper ones (such as $x \times x$)

During this stage, as it was not an actually progressive task, it was impossible to check the state of the kernel at each moment. Therefore, until most of the modifications were done, there was no way of knowing if the work done was correct. In the end the new implementation proved correct, and the execution time decreased many times-fold. Some minor modification were further added, which reduced even more the execution time.
Finally, the implementation obtained performed just as well as the "trivial" one, consuming much less time.

All the work done until that moment aimed at obtaining a single layer arc-cosine kernel, while the project aimed at building a Deep Kernel Machine. In this case, the "trivial" implementation was obviated, and, instead, the optimizations were directly designed. Once this process was considered done, and a seemingly correct implementation was obtained, it was tried using a single layer, and when its results proved correct, it was tried with more layers. In every of them, precision was checked, in order to discard errors in higher layers. In the end, a highly efficient arc-cosine was obtained, capable of working as a single layer SVM, or to use as many layers as desired.
5 Experiments

5.1 Experiment description

In order to evaluate the performance of the obtained program, some experiments had to be developed. These experiments aimed to obtain the precision and speed of the code, and in order to do so, it had to be tried in, at least, a problem. The chosen problem was the *MNIST* dataset.

5.2 *MNIST* dataset

One of the most common problems in Machine Learning is the one contained in the *MNIST* dataset [7], and, therefore, is the one that has been chosen to work in this project. The mentioned problem is the one consisting in recognizing handwritten digits. The whole *MNIST* dataset contains 60,000 images to train the model, as well as 10,000 additional images to test the obtained model (further explanation in 5.3. The images represent handwritten digits ranging from 0 to 9. These images have a 28x28 pixels format, in which each pixel represents the intensity of the black color in it, ranging from 0 to 255.

The training and the testing images were obtained from different sources, as the person who wrote the digits in the training set was different than the one who wrote the images in the testing set. In the figure 7 there are examples of these numbers.

Due to its frequent usage, *MNIST* is considered an *standard bench*
mark in the area of Machine Learning [10, 12], which is one of the main reasons of its choice. In table 1 there are the reference values for this dataset, extracted from [13].

<table>
<thead>
<tr>
<th>Method</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM-rbf</td>
<td>1.6</td>
</tr>
<tr>
<td>Neural Network</td>
<td>1.9</td>
</tr>
<tr>
<td>DBN, [30]</td>
<td>1.25</td>
</tr>
<tr>
<td>Stacked Autoencoder, [29]</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 1: Results obtained in MNIST by different Machine Learning problems

The use of libsvm forced us to prepare the input files in a certain way, as it only recognizes them in that format. This format specified that the input file had to have a line for every instance, and every instance had to have the following form:

\[
\text{label } 1:\text{feature}_1 2:\text{feature}_2 \ldots M:\text{feature}_m
\]

being label the known value for that instance in case of supervised learning problems, or any desired value in case of unsupervised learning (the label is ignored in unsupervised learning). On the other hand, feature\_n represents the value of the nth feature in that instance.
In order to do so, a short program was developed that converted the MNIST input files into the desired format.

5.3 Results

Once the dataset was in a readable format, the experiments started. At this point it is necessary to add a further explanation on how Machine Learning programs are trained. As it has already been explained, Machine Learning programs use sets of data, called datasets, in order to learn on their own. However, not all the available data is directly provided to the program for it to learn. Instead, the dataset is divided into the training, the validation and the testing sets, each of them containing a part of the total instances. The choice of this sets is sometimes difficult, but necessary, and the reason for this are two phenomenas called overfitting and underfitting:

- Underfitting: It appears when the model used in the training of the data is not complex enough to represent this data and, therefore, it is incapable of extracting any useful information. Used in further data will result in extremely bad predictions.

- Overfitting: On the contrary, overfitting appears when the model used is too complex for the data being used. Even though at first sight it would appear as a good result, actually, is extremely undesirable. The reason for this is that if a model fits too precisely the data, it is likely to perform badly when applied to instances from outside the initial set of data. Therefore, it is important for
Experiments

a model to adjust the data, but at the same time maintaining a certain degree of generalization

In order to illustrate both cases, in the figure 8 there are examples for them, and an example of a correctly fitted model.

Figure 8: Left: underfitted model | Center: correctly fitted model | Right: overfitted model [21]

In the figure each dot corresponds to an instance of the problem being analyzed, and the line represents the function learnt by the program to represent their behaviour. As it can be seen, the underfitted model is far from correctly representing the model, as most of the points are far from the line. On the other hand even though in the overfitted model each point is exactly on line, it is obviously not representing the general behaviour of the data. Finally, the correctly fitted model, even though is not so precise as the overfitted one, is likely to perform better than this one when faced with new instances.

The explanation of the overfitting and the underfitting is necessary to understand the need for three separated datasets. Basically, the first set, the training one, is provided to the program in order for it to learn. The model obtained by this one is then tried with the instances of the
Experiments

Implementation and study of a deep and wide Kernel Machine

validation set. The same process is repeated with all the parameter combinations desired, training a model and then validating it. The one with the lower error in the validation set is the one which is better fitted, as it has avoided both underfitting and overfitting. Therefore, it shows the optimal choice of parameters for that method and problem. Finally that combination of parameters is trained with both training and validation sets, and it is later tested in the test set, which is like a final security measure to avoid an overfitting of the validation set. If the error is low enough, it is considered to be a good model. In case the error is too high, the good results in validation are irrelevant and a new process must be started.

In our case, the MNIST dataset already provides a separated test set, but the other set had to be split into training and validation. The choice done consisted in taking 50000 instances as training set and 10000 as validation set, which were randomly chosen. Fortunately, the MNIST dataset contains a reasonably high number of instances, which allows for good sizes in every set.

The parameters that had to be tried were the number of layers, the degree \( n \) and the C parameter of the SVM-vector, which indicates the tolerance to errors, the higher the C the less tolerant with misclassifications the model is. The values for each parameter tried were:

- Num. Layers: from 1 to 6, increased by unit. This values are the same tried in [1]

- Degree: 0 and 1. The intention was to try also the 2nd degree,
Experiments

and, even more, the formula for the 3rd degree was also calculated. However, when tried in the MNIST dataset with more than one layer, the value of $k(x, y)$ resulted to be null. The reason for this was the fact that the value obtained was so large that it overflowed the double in which it was stored. Therefore, only 0 and 1 values were tried. Being as this is a project based mostly in the development of the student, it was not seen as a major drawback. However, in case the project were to continue, this is one of the most urgent things to fix.

- C: The values for the C parameter ranged from $10^{-2}$ to $10^5$, increasing in $\sqrt{10}$ at each step.

As it has already been explained, Machine Learning in general have high computational requirements. In this particular case, this requirements were not excessively high, comparing with some of the most costly existent algorithms, but they were neither cheap. More or less, training the model with the 50000 training images would take approximately a day in the laptop in which the project was being performed. Therefore, trying all the possible combinations in it would take an impractical amount of time. In order to be able to perform them, the Computer Science department (CS) [26, 27] of the UPC provided access to a computing cluster [28], which allowed for this experiments to be run much faster and in parallel, which greatly reduced the required time.
The results obtained in the validation set for each of combination can be found in the next table:

<table>
<thead>
<tr>
<th>Num. Layers</th>
<th>Degree</th>
<th>C</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$10^6$</td>
<td>2.6496</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$10^{-2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0.01</td>
<td>2.4996</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3000</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>3000</td>
<td>2.7995</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3000</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>$10^6$</td>
<td>3.1995</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3000</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1000</td>
<td>3.6494</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3000</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>1000</td>
<td>4.016</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3000</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2: Results obtained in validation set

In order for the table not to be too large, only the best C for each combination of number of layers and degree is shown.
5.4 Best results

As it can be seen, all combinations using the first degree performed equally well. Because of this, each of this combination was trained in the whole dataset and tried in the test set, with the following results:

<table>
<thead>
<tr>
<th>Num. Layers</th>
<th>Degree</th>
<th>C</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10^{-2}</td>
<td>3.68</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3000</td>
<td>1.93</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3000</td>
<td>1.75</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3000</td>
<td>1.71</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3000</td>
<td>1.59</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3000</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Table 3: Results obtained in test set

So, in the end, the best found combination was using 5 layers with a degree equal to one, resulting in an error of: 1.59
6 Conclusions

It is interesting to notice the tendency of the error 3, as it tends to decrease as the number of layers increases. The same could be said with the degree, as the first degree has obtained much better results than the zeroth. From these two facts combined a conclusion can be extracted: the more complex the model, the better the results are likely to be. That being said, however, it is important to point out that these results are not in any way exhaustive. In order for these results to be representative, more experiments should be done, increasing the number of layers and, specially, the degrees used, as well as trying the model in other problems.

It should be also pointed out the fact that the results does not suffer of great variability from one parameter combination to another. This, together with the comparatively low number of parameter combinations, implies that a solution close to its optimal is quick to find.

Therefore, this method requires neither a difficult tuning in order to perform well, nor high execution times. These two facts combined make it fast to train compared with other Deep Learning methods. However, as a side effect, the low variability of results between parameter combinations implies that if the method does not perform well in a problem, it is difficult to greatly improve its performance.

In relation to whether the goals of the project have or not been fulfilled, the agreement is that they have been so. The reasons to consider so are, basically:
Conclusions

1. A functional Deep Kernel Method has been implemented, with an acceptable performance both in precision and in execution time.

2. The method has been tried in the MNIST dataset and, even though the analysis has not been as exhaustive as it could have been, due to time restrictions, it has allowed for some conclusions to be extracted.

3. By means of the kernel compositions, new options have been proposed, which ought to be further studied in future work.

To conclude, the project is considered to have been successfully accomplished, while leaving some pending tasks to be done in further projects.
7 Bibliography


[2] Ludovic Arnold\(^1\), Sébastien Rebecchi\(^1\), Sylvain Chevallier\(^1\), Hélène Paugam-Moisy\(^{1,3}\) *An Introduction to Deep Learning*, Université Paris-Sud\(^1\), LIMSI\(^2\), Université Lyon\(^3\)


Bibliography


Bibliography


Bibliography


[29] P. Lamblin, Y. Bengio Important gains from supervised netuning of deep architectures on large labeled sets