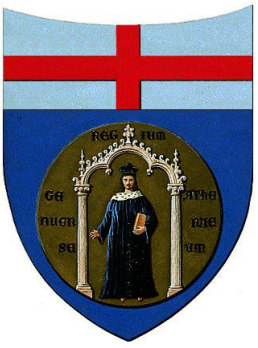
UNIVERSITY OF GENOA



Machine Learning in Resource-constrained Devices: Algorithms, Strategies, and Applications

by

Edoardo Ragusa

A thesis submitted in partial fulfillment for the degree of
Doctor of Philosophy in Science and Technology for Electronic and Telecommunication Engineering,
Curriculum: Electromagnetism, Electronics, Telecomunications

in the
Faculty of Engineering
Department of naval, electric, electronic and telecommunications engineering

Tutors: Prof. Paolo Gastaldo, Rodolfo Zunino
Coordinator of PhD Course: Prof. Mario Marchese

February 2019

Declaration of Authorship

I, Edoardo Ragusa, declare that this thesis titled, 'Machine Learning in Resource-constrained Devices: Algorithms, Strategies, and Applications' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

'With four parameters I can fit an elephant, and with five I can make him wiggle his trunk."

John Von Neumann

UNIVERSITY OF GENOA

Abstract

Faculty of Engineering
Department of naval, electric, electronic and telecommunications engineering

Doctor of Philosophy in Science and Technology for Electronic and Telecommunication Engineering,
Curriculum: Electromagnetism, Electronics, Telecomunications

by Edoardo Ragusa

The ever-increasing growth of technologies is changing people's everyday life. As a major consequence: 1) the amount of available data is growing and 2) several applications rely on battery supplied devices that are required to process data in real time. In this scenario the need for ad-hoc strategies for the development of low-power and low-latency intelligent systems capable of learning inductive rules from data using a modest amount of computational resources is becoming vital. At the same time, one needs to develop specific methodologies to manage complex patterns such as text and images.

This Thesis presents different approaches and techniques for the development of fast learning models explicitly designed to be hosted on embedded systems. The proposed methods proved able to achieve state-of-the-art performances in term of the trade-off between generalization capabilities and area requirements when implemented in low-cost digital devices. In addition, advanced strategies for efficient *sentiment analysis* in text and images are proposed.

Acknowledgements

The work presented in this Thesis was possible thanks to the help of many people.

First of all, I would like to thank my parents and the rest of my family for the sustain they gave me every day.

I would like to thank Professors Paolo Gastaldo and Rodolfo Zunino; without their help and contributions all this work would not have been possible. I would also thank all the past and present members of Sealab and SenticTeam. Each one of them helped me in my work and taught me valuable lessons.

I also want to express my appreciation for my friends and my brother that always remind me of what is really important in life.

Finally, I heartily thank my girlfriend Valentina for all the time we've spent together. I can never put into words how much you mean to me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine Learning [4] is a fascinating interdisciplinary field where statistics, geometry, algebra and probability theory merge together. The purpose of Machine Learning techniques is to infer properties on unseen data given a previous learning or training stage, during which the relationship f between the data and the properties to be inferred is derived. In other words, machine learning supports systems that can abstract knowledge from data rather than simply memorize a set of rules for labeling them.

The word learning stands for an algorithmic procedure by which, from a finite number of examples, the inductive rule f is obtained. The selection of the most profitable training approach is influenced by multiple factors, including but not limited to, feature extraction process, convergence speed, convexity, computational cost and memory requirements. A learning algorithm defines the quality of rule f in term of generalization ability but also in term of the computational cost of the eventual predictor. The latter aspect depends on the structure of the hypothesis space and by the compression ability of the training procedure.

Nowadays the development of pervasive electronics enforces the requirement of procedures that are capable of balance computational cost of the training phase, with efficiency of the eventual predictor. In this scenario, it is necessary to recalibrate the learning phase to minimize the number of computations required. In some cases, the changes start from the hypothesis space that characterizes the eventual predictor. In these scenarios, the hypothesis spaces are subject to constraints with the goal of promoting the digital implementations of the classifier. In the following, these hypothesis spaces are called hardware-friendly. Unfortunately, there is a sort of trade-off between training cost and efficiency of the predictor.

Learning models based on randomness and deep learning approaches represent the two extreme cases. In fact, the random based ones discard some portions of the training phase in favour of a lower number of computations without losing generalization capabilities of the models. As a major result, the learning procedure can be performed with a modest number of floating point operations, but the eventual predictors suffer from relatively low accuracy and modest performance in term of compactness respect to fully trained ones.

On the other hand, deep learning techniques have obtained astonishing results compared to traditional machine learning models in many applications, mostly thanks to the ability of automatically extracting complex features sets from the data. This abstraction ability comes at expense of an increased computational effort that makes the deployment of deep architectures an important research topic.

This Thesis assesses the problem of learning in resources constrained scenario from two points of view. Firstly, the focus is put on single hidden layer feed forward neural networks (SLFNs) trained using random based approaches. In this phase, the goal is the development of machine learning methods based on hardware friendly hypothesis spaces that require a light learning phase. This research line is concluded by the proposal of architectures for the digital deployment of predictors in low-end digital devices. Secondly, the problem of complex feature extraction is explored, providing insight about possible strategies that can fasten training approaches exploiting domain knowledge and transfer learning. In particular, tasks related to sentiment analysis in text and images are addressed.

The Thesis is organized as follows: Chap. 2 discusses fast learning based on single hidden layer hypothesis space; Chap. 3 refers to the results achieved extending a subset of the approaches presented in Chap. 2 to the case of tensor input domains. Chapter 4 describes the ad-hoc digital architectures designed for the deployment of eventual predictor trained following procedures described in Chap. 2. Part of the discussed techniques are tailored to the task of subjectivity detection in text in Chap. 5. Finally, a study about the impact in term of computational efficiency and accuracy of deep learning architectures for the task of image polarity detection is proposed in Chap. 6.

1.1 Contribution

The contributions of this Thesis can be summarized in the following points: contributions regarding random basis networks (Chapters 2, 3, 4), and application to information

retrieval and sentiment analysis (Chapter 5, 6). The contribution of the first part can be summarized in three main points:

- An analysis about the role of randomization in the training process of a learning machine, and about the affinities between two well-known schemes, namely, Extreme Learning Machines (ELMs) and the learning framework using similarity functions is provided [5–7]. These paradigms share a common approach to inductive learning, which combines an explicit remapping of data with a linear separator; however, they seem to exploit different strategies in the design of the mapping layer. The research shows that, in fact, the theory of learning with similarity functions can stimulate a novel interpretation of the ELM paradigm, thus leading to a common framework. New insights into the ELM model are obtained, and the ELM strategy for the setup of the neurons' parameters can be significantly improved. Experimental results confirm that the novel methods outperform conventional approaches, especially in the trade-off between classification accuracy and machine complexity (i.e., the dimensionality of the remapped space).

- Machine learning algorithms are typically designed to deal with data represented as vectors. Several major applications, however, involve multi-way data. In those cases, tensors endow a more consistent way to capture multi-modal relations. This Thesis presents a tensor-oriented machine learning framework, and shows that the theory of learning with similarity functions provides an effective paradigm to support this framework [8]. The performance of the tensor-based framework is evaluated on a set of complex, real-world, pattern-recognition problems. Experimental results confirm the effectiveness of the framework, which compares favorably with state of the art machine learning methodologies that can accept tensors as inputs. Indeed, a formal analysis proves that the framework is more efficient than state of the art methodologies also in terms of computational cost. The Thesis thus provides two main outcomes: (1) a theoretical framework that enables the use of tensor-oriented similarity notions and (2) an efficient notion of similarity that leads to computationally efficient predictors.

- The availability of compact digital circuitry for the support of neural networks is a key requirement for resource constrained embedded systems. This Thesis tackles the implementation of single hidden-layer feed-forward neural networks, based on hard-limit activation functions, on reconfigurable devices [9, 10]. The resulting design strategies rely on a novel learning procedure that inherits the approach discussed in Chapter 2. Experimental tests confirm that the design approach leads to efficient digital implementations of the predictor on low-performance devices

that performs favorably in term of area occupation respect to state-of-the-art approaches.

The second part (Chapter 5 and 6) provides three main results:

- Subjectivity detection is a task of natural language processing that aims to remove 'factual' or 'neutral' content, i.e., objective text that does not contain any opinion. Such a pre-processing step is crucial to increase the accuracy of sentiment analysis systems, as these are usually optimized for the binary classification task of distinguishing between positive and negative content. This Thesis extends the ELM paradigm to a novel framework that exploits the features of both Bayesian networks and fuzzy recurrent neural networks to perform subjectivity detection [11]. In particular, Bayesian networks are used to build a network of connections among the hidden neurons of the conventional ELM configuration in order to capture dependencies in high-dimensional data. Next, a fuzzy recurrent neural network inherits the overall structure generated by the Bayesian networks to model temporal features in the predictor. A formal analysis proves that the framework is more efficient than state-of-the-art in terms of computational cost. Finally, experimental results confirmed the ability of the proposed framework to deal with standard subjectivity detection problems.

- The availability of an effective embedding for textual information is a primal challenge in commonsense reasoning, because it is at the basis of the whole processing flow and it strongly influences the quality of the entire analysis. In this Thesis, a recently introduced technique for finding cognitively meaningful paths is applied to AffactiveSpace, a multidimensional space of commonsense knowledge primarily used for sentiment analysis. The proposed protocol provides engineers and data scientists with a qualitative measure of concepts distributions in a graphical format that enables the analysis of embedding properties and thus it is useful to evaluate and optimize the embedding space itself. Experimental section involved the characterization of AffectiveSpace proving that the proposed approach can be effectively used to describe embeddings; further it is shown how data displacement in this specific embedding is coherent with the hourglass model of emotions.

- Deep convolutional neural networks (CNNs) provide an effective tool to extract complex information from images. In the area of image polarity detection, CNNs are utilized in combination with transfer learning techniques. Thus, polarity predictors in general exploit a pre-trained CNN as feature extractor that in turn feeds a classification unit. While the latter unit is trained from scratch, the pre-trained CNN is subject to fine tuning. Such generic framework is at the base of almost

all the state-of-the-art models for polarity detection on images. The convolutional neural network employed as feature extractor strongly affects the performance of the model. The Thesis analyzes the state-of-the-art about image polarity detection and enlighten that a fair comparison between different models is difficult due to the diverse architecture proposed in different works. The performances of these architectures are then compared by defining an experimental protocol that allows a fair comparison between existing convolutional neural networks. The performances are evaluated both in terms of generalization abilities and in terms of computational complexity. The latter attribute becomes critical when considering that polarity predictors -in the era of social network and custom profiles- might need to be updated within a short time interval (i.e., hours or even minutes) using limited amount of computational resources. As a major consequence, such predictors should properly address the trade-off between classification accuracy and computational load. In this regard, the Thesis provides practical hints on advantages and disadvantages of the examined architectures.

Chapter 2

Fast Learning Approaches

The availability of a large quantity of data creates new opportunities and new challenges in the area of statistical learning [12–14]. Such availability boosted many strategic domains such as business intelligence [15], Internet of Things (IoT) [16], Natural Language Processing (NLP) [11, 17], and social media monitoring [18]. Indeed, both academia and industry have been investing resources and money in the development of new methodologies that can cope with the problem of extracting information from available sources, both from a theoretical point of view [12, 14] and from an implementation point of view [19, 20].

As an example, deep learning architectures [14] have proved able of achieving outstanding results in term of generalization ability. On the other hand, these architectures involve a few major issues: 1) the implementation of both learning phase and inference phases [21] require high performance hardware; 2) the training process is extremely sensitive to parameterization; finally, 3) to train a deep network a big dataset should be available.

Computational aspects, though, affect also standard machine learning approaches, which in general may deal with medium-size training set (i.e., thousands of samples). This issue becomes prominent when the training procedure should run on resource-constrained device, e.g., electronic embedded systems. Under this scenario, one often needs to rely on a paradigm that can address the ever-present trade-off between predictor complexity and generalization ability. Toward that end, paradigms combining single-hidden-layer feedforward networks (SLFNs) and random basis functions achieved significant results, because 1) they require modest computational resources [9], 2) the convex nature of the optimization problem simplifies the implementation phase [13], and 3) the limited amount of free parameters involved allow one to train these models also when small datasets are available. Random Radial Basis Functions [22], Random Vector Functional-Link (RVFL) [23], Extreme Learning Machines (ELMs) [24, 25], and Weighted Sum of

Random Kitchen Sinks [26] represent very interesting instances of such approach. These models all share the common idea of initializing the hidden layer by using randomization to support a fixed transformation of input data. As a result, the learning process should only adjust a linear separator in the upper layer (i.e., in the remapped space). Remarkably, this simplification does not affect the universal approximation abilities of the resulting machines [13]. Between these paradigms, ELM is the one that presents the most solid theoretical background and most of the literature refers to random based neural network as ELM. For these reasons, in the following, we will refer to random based models as ELM or random basis model interchangeably.

This chapter aims to open new vistas on the above learning approaches, by proving that there exists a parallelism between random basis functions and similarity functions [27]. In this regard, the first contribution provided by the present work is a novel analysis of the ELM hypothesis space that takes advantage of the convergences between ELM and the theory of learning with similarity functions (SIM). Accordingly, this research shows that the standard ELM model implements a sort of speculative policy in data mapping. In practice, each hidden neuron remaps an input datum irrespectively of the others; more importantly, the basis function acts as similarity function that remaps input data according to a random similarity notion with respect to a random reference point (landmark). As a major consequence, while the mapping layer does not require any parameter-fitting, several neurons might cover irrelevant portions of the data space or be badly parameterized, thus compromising efficiency at capturing the structure of the problem at hand. This is even more important when considering that the computational complexities of both training and run-time operations depend on the size of the mapping layer. Therefore, this research demonstrates that in principle the ELM framework provides a broader hypothesis space with respect to the standard theory of similarity functions. However, this comes at the expense of a small deterioration of the learning bounds.

The second contribution provided by the present chapter is a set of novel training strategies for the ELM model. The eventual goal is to overcome the limitations brought about by the standard speculative policy that drives the setup of the mapping layer. The proposed training strategies still rely on random landmarks; conversely, the configuration parameter of the activation/similarity function are set based on selection procedures that mitigate the presence of pathological configurations. Indeed, such task is accomplished by avoiding standard, computationally demanding strategies such as back propagation or model selection.

This research actually shows that the theory of learning with similarity functions may inspire heuristics, yet valuable approaches for selecting the most promising value for the

shape parameters of activation functions.

The chapter is structured as follows: in the first two sections, the selected paradigms are presented. Details about the affinities between ELM and SIM paradigms are deeply investigated in section 2.3. Following, in section 2.4, the geometrical consistence between the selected activation function and hidden parameter sampling is discussed. Finally, section 2.5 summarize the algorithms derived from the proposed analysis and presents the experimental results that empirically validates their effectiveness.

2.1 Random Feed Forward Neural Networks

Feed forward neural networks [28] represent one of the most widely used learning paradigms thanks to the excellent trade-off between computational load and the generalization capabilities of the eventual predictor. Given an input datum $\boldsymbol{x} \in \mathbb{R}^D$, the general structure of a SLFN is:

$$f(\boldsymbol{x}) = \sum_{i=1}^{N} \beta_j h_j(\boldsymbol{x}, \boldsymbol{\omega}_j, \sigma_j) \tag{2.1}$$

where N is the number of hidden neurons, h_j is a nonlinear activation function, $\boldsymbol{\omega}_j$ are the hidden parameters, σ_j is the shape parameter of the j-th neuron and $\boldsymbol{\beta}$ is the set of output weights.

Given a set of labeled data $\mathcal{T} = \{(\boldsymbol{x}_i, y_i), i = 1, ..., Z, \boldsymbol{x}_i \in \mathbb{R}^D, y_i \in [-1; 1]\}$, training of such models consists in tuning the network's parameters to minimize a loss function L:

$$\min_{\boldsymbol{\beta}, \boldsymbol{\sigma}, \boldsymbol{\Omega}} L(\mathcal{T}, f(\boldsymbol{\beta}, \boldsymbol{\Omega}, \boldsymbol{\sigma})) \tag{2.2}$$

In practice, training consists in the solution of an optimization problem that, based on the specific loss function L and the predictor f, can present different levels of complexity.

Since its proposal in 1986, back propagation algorithm (BP) [29] is at the base of the most effective and used procedures for the training of neural networks. This optimization technique is based on an iterative update of the models parameters $\{\boldsymbol{\beta}, \boldsymbol{\sigma}, \boldsymbol{\Omega}\}$ in the opposite direction with respect to the gradient of the loss function. This optimization technique ensures globally optimal solution only when both L and f are convex respect to the tuned parameters. Despite its effectiveness, application of BP in SLFN training suffers from slow convergence and local minimum problems because f is not convex by construction. The literature provides plenty of approaches that alleviate this issue

[30, 31] but in general sets of hyper-parameters are introduced making the tuning process challenging. Furthermore, the requirements imposed by the gradient calculation limit the allowed activation functions h to the differentiable ones.

A widespread strategy to tackle the aforementioned problems is based on the so-called random models [22–24, 26] . The common idea behind all these paradigms consists in performing a random nonlinear transformation of the input data. This is achieved by setting the parameters $\boldsymbol{\Omega}$ and $\boldsymbol{\sigma}$ randomly. This design choice affects dramatically the computational load of the training phase because f becomes convex with respect to the training parameters. As a consequence, the optimization problem turn into to the setup of a linear separator $\boldsymbol{\beta}$ in the new space. Given that f is convex, if the loss function is also convex, the optimization problem admits a globally optimal solution. Between the convex loss functions means square error (MSE) is the most used in literature:

$$\min_{\boldsymbol{\beta}}\{\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta}\|^2\} \tag{2.3}$$

where $\boldsymbol{H}$ denote a $Z \times N$ matrix, with $h_{ij} = h_j(\boldsymbol{x}_i, \boldsymbol{\omega}_j, \sigma_j)$;

The success of MSE is strictly related to the fact that, the resulting optimization problem, not only becomes convex, but it also admits a closed form solution:

$$\boldsymbol{\beta} = \boldsymbol{H}^{-1}\boldsymbol{y} \tag{2.4}$$

Interestingly, universal approximation capability [32] is maintained and the set of admissible activation functions h is extended to a broader set with respect to the ones allowed from BP [13]. Numerical stability issue and overfitting affect the solution of this learning problem. Fortunately, the theory derived in [33] proves that regularization techniques can further improve the approach generalization performance, at the same time, numerical stability issue can be limited by smoothing the solution. As a result, it is convenient to replace cost function (2.3) with:

$$\min_{\boldsymbol{\beta}}\{\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|^2\} \tag{2.5}$$

where λ is the regularization parameter that controls the ratio between data fitting and the smoothness of the solution. This regularized version (RMSE) of the loss function still admits an analytical closed form solution. When $Z \leq N$, one has:

$$\boldsymbol{\beta} = \boldsymbol{H}^T(\lambda\boldsymbol{I} + \boldsymbol{H}\boldsymbol{H}^T)^{-1}\boldsymbol{y} \tag{2.6}$$

conversely, when $Z > N$ one has:

$$\boldsymbol{\beta} = (\lambda \boldsymbol{I} + \boldsymbol{H}^T \boldsymbol{H})^{-1} \boldsymbol{H}^T \boldsymbol{y} \tag{2.7}$$

It is important to stress the fact that, despite the closed form solution shown in equations (2.4) and (2.5), the efficient computation of the solution does not involve the explicit matrix inversion, but relies on algorithms for the solution of linear equation systems that are computationally more efficient [34]. Interestingly, the regularized mean square error is not the only convex loss function available to tune the linear separator; in [35] the authors compared different loss functions in term of generalization ability of the eventual predictor and the results proved that RMSE could not be the best choice in some cases. Despite this last observation, RMSE boasts a solid and well-structured literature with optimized solvers that enables efficient implementation in resource constrained scenario. For this reason, in the following, RMSE loss function is considered.

Finally, to complete the discussion about ELM the complete training procedure is presented in Algorithm 1. Notably, the training procedure can be summarized in a few lines of code.

Algorithm 1 The learning scheme based on random hidden layer

Input

- a labeled training set $\mathcal{T} = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z\}$
- a set of activation functions $\boldsymbol{h}$
- number of neurons N

0. **Initialize**
extract the random set $\{\boldsymbol{\omega}_i, \sigma_i\}, i = 1, ..., N$
1. **Mapping**
remap all the patterns $\boldsymbol{x} \in \mathcal{T}$ by using the following mapping function

$$\boldsymbol{\phi}(\boldsymbol{x}) = \{h_1(\boldsymbol{x}, \boldsymbol{\omega}_1, \sigma_1), ..., h_N(\boldsymbol{x}, \boldsymbol{\omega}_N, \sigma_N)\}$$

2. **Learning**
train a linear predictor in the space $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^N$

2.2 Theory of Learning with Similarity Functions

The theory of learning with similarity function presented in [27] consists of a theoretical framework that extends the mathematical paradigm of Kernel learning [36]. This general

theory sets the sufficient conditions for a similarity function to allow one to "learn well", without requiring the function of being semi-definite positive or even symmetric.

The following pair of definitions summarizes the crucial elements of the theoretical framework; the first one addresses the notion of pairwise similarity function, K, while the second introduces the most basic, yet intuitive notion of good similarity function.

Definition 1. [27] A similarity function over $\mathcal{X}$ is any pairwise function $K : \mathcal{X} \times \mathcal{X} \rightarrow [-1, 1]$. K is defined as a symmetric similarity function if $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}, K(\boldsymbol{x}, \boldsymbol{x}') = K(\boldsymbol{x}', \boldsymbol{x})$.

Definition 2. [27] K is a strongly (ϵ, γ)-good similarity function for a learning problem $\boldsymbol{P}$ if at least a $(1 - \epsilon)$ probability mass of examples $\boldsymbol{x}$ satisfy:

$$E_{\boldsymbol{x}' \sim \boldsymbol{P}}[K(\boldsymbol{x}, \boldsymbol{x}')|y(\boldsymbol{x}) = y(\boldsymbol{x}')] \geq E_{\boldsymbol{x}' \sim \boldsymbol{P}}[K(\boldsymbol{x}, \boldsymbol{x}')|y(\boldsymbol{x}) \neq y(\boldsymbol{x}')] + \gamma \tag{2.8}$$

The definition of (ϵ, γ)-good similarity function imposes a set of stringent assumptions that are not applicable to the real word scenario if K is fixed a-priori.

An (ϵ, γ)-good similarity function satisfies less stringent constraints (again inherited from [27]):

Definition 3. [27] A similarity function, K, is an (ϵ, γ)-good similarity function for a learning problem $\boldsymbol{P}$ if there exists a bounded weighting function ω over $\mathcal{X}(\omega(\boldsymbol{x}') \in [0, 1]))$ for all $\boldsymbol{x}' \in \mathcal{X}$ such that at least a $(1 - \epsilon)$ probability mass of examples $\boldsymbol{x}$ satisfy:

$$E_{\boldsymbol{x}' \sim \boldsymbol{P}}[\omega(\boldsymbol{x}')K(\boldsymbol{x}, \boldsymbol{x}')|y(\boldsymbol{x}) = y(\boldsymbol{x}')] \geq E_{\boldsymbol{x}' \sim \boldsymbol{P}}[\omega(\boldsymbol{x}')K(\boldsymbol{x}, \boldsymbol{x}')|y(\boldsymbol{x}) \neq y(\boldsymbol{x}')] + \gamma \tag{2.9}$$

The above definition uses a weighting function, ω, to balance the relative significance of each sample, $\boldsymbol{x}'$. When considering the eventual learning algorithm, the definition requires that a bounded weighting function, ω, exists, although this does not imply that such a function is known *a-priori*. In other terms, any similarity expression is, in principle, a (ϵ, γ)-good similarity function, and one needs a suitable criterion to find the weighting function ω that minimizes ϵ and maximizes γ.

In practice, one should replace the expectation with an average over a set of landmarks, i.e., the examples $\boldsymbol{x}'$. As a result, the weighting function ω will satisfy Definition 3 in correspondence of the landmarks at-hand. The outcome of this observation is that the notion of good similarity function allows one to setup a learning scheme based on a hypothesis space. To this purpose, one needs

Algorithm 2 The learning scheme that exploits the theory of learning with (ϵ, γ)-good similarity functions

Input

- a labeled training set $\mathcal{T} = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z\}$
- a similarity function K
- number of landmarks L

0. **Initialize**
extract L random samples $\mathcal{L} = \{\boldsymbol{l}_n; n = 1, .., L\}$ from $\mathcal{T}$

1. **Mapping**
remap all the patterns $\boldsymbol{x} \in \mathcal{T}$ by using the following mapping function

$$\boldsymbol{\phi}(\boldsymbol{x}) = \{\frac{1}{\sqrt{L}} K(\boldsymbol{x}, \boldsymbol{l}_1), ..., \frac{1}{\sqrt{L}} K(\boldsymbol{x}, \boldsymbol{l}_L)\}$$

2. **Learning**
train a linear predictor in the space $\boldsymbol{\phi} : \mathcal{X} \to \mathbb{R}^L$

- L landmarks, i.e., a subset of the original dataset which is randomly drawn from the domain distribution $p(\mathcal{X})$ that characterizes P. Both labeled and unlabeled patterns provide an admissible source of landmarks.
- A similarity function K.

To build the hypothesis space, the domain space $\mathcal{X}$ is first remapped into a new space $\mathbb{R}^L$. Accordingly, for every pattern, $\boldsymbol{x}$, one computes the similarities, K, between $\boldsymbol{x}$ and each landmark. In the second step, a linear predictor is trained in the new space, $\mathbb{R}^L$. The eventual hypothesis space can be formalized as

$$f(\boldsymbol{x}) = \sum_{j=1}^{L} \omega_j K(\boldsymbol{x}, \boldsymbol{l}_j) \tag{2.10}$$

where the weights, ω_j, are computed by adjusting a linear predictor. Algorithm 2 outlines the associate learning procedure.

Algorithm 2 relies on the similarity function K to remap the original space into a new space where data are separated by a (possibly large) margin with error, ϵ. Then, the task of tuning the weighting function, ω, is assigned to the linear predictor. The learning abilities of this procedure have been formally analyzed in [27]: if one set $L = 16 \cdot ln(4/\epsilon*)/\gamma^2$, then with probability at least $(1 - \epsilon^*/2)$ there exists a low-error $(\leq \epsilon + \epsilon^*)$, large-margin $(\geq \gamma/2)$ separator in the feature space.

2.3 Convergence between Learning with Similarity Functions and Extreme Learning Machine

Several convergences exist between the predictors derived by the paradigms introduced in sections 2.1 and 2.2 and their learning procedures [6, 9, 37]. In fact, both approaches presented in algorithms 1 and 2 relies on a two step procedure: firstly, data are projected in a remapped space with explicit dimension, L for SIM paradigm and N for ELM. The second step of the procedure instead involves the tuning of a linear separator in the remapped space.

In principle, all the differences consist of the different remapping strategies employed, however, a careful analysis unveils that many common points exist. Understanding these similarities can stimulate new ideas and developments in the field, for these reasons, in the following, ELM hypothesis space will be reinterpreted in view of learning with similarity function. It is important to note that the proposed interpretation is not the only admissible one, but offers a different point of view on the learning abilities of the model with respect to previous works [33, 38, 39].

In general, such convergences become evident when the ELM model utilizes activation functions that can be reinterpreted as similarity functions [37]. Thus, let $\phi(\boldsymbol{x}, \boldsymbol{r}_j, \chi_j)$ be parametric activation/similarity function that remaps $\boldsymbol{x}$ into $\mathbb{R}$ by using a landmark $\boldsymbol{r}_j \in \mathbb{R}^D$ as a reference point; χ_j is the configuration parameter for ϕ. As a result, the hypothesis space (2.1) and the hypothesis space (2.10) may represent two instances of the following general hypothesis space:

$$f(\boldsymbol{x}) = \sum_{j=1}^{N} \beta_j \phi(\boldsymbol{x}, \boldsymbol{r}_j, \chi_j) \tag{2.11}$$

From the viewpoint of learning with similarity functions (2.10), the vectors $\boldsymbol{r}_j$ in equation (2.10) embeds data belonging to the (unknown) distribution $\boldsymbol{P}$. Besides, $\chi_1 = \chi_2 = ... = \chi_N$, as the hypothesis space (2.10) does not admit multiple configurations for the similarity function. In the case of the ELM hypothesis space (2.1), conversely, both the landmarks $\boldsymbol{r}_j$ and the parameters χ_j are selected randomly. Therefore, the vectors $\boldsymbol{r}_j$ can embed any datum belonging to $\mathbb{R}^D$ and the similarity function adopts a different parameterization for each landmark.

One of the contribution of this Chapter is to show, formally, that the connection between the two paradigms is tight. The novel insights on the hypothesis space can be stimulated when considering the parallelism between such hypothesis space and the hypothesis space (2.1). In the following, subsections 2.3.1 and 2.3.2, respectively, will show that -in

principle- the ELM model can inherit the learning bounds of the hypothesis space (2.1) even if 1) $\chi_1 \neq \chi_2 \neq ... \neq \chi_N$, and 2) $\boldsymbol{r}_j \in \mathbb{R}^D$.

2.3.1 Shape Parameter of the Activation Function

The hypothesis space set by the theory of learning with similarity functions in principle does not admit a configuration parameter for the similarity function at-hand. Hence, in the case of a similarity function belonging to a parametric family, one should implicitly set $\chi_1 = \chi_2 = ... = \chi_N = \chi$, where χ is a predetermined value. On the other hand, the ELM model randomly sets each χ_j. Theorem 2.1 indeed proves that the ELM mapping layer can support a low error ϵ large margin γ linear separator:

Theorem 2.1. *Consider a set of N landmarks $\{\boldsymbol{r}_1, \boldsymbol{r}_2, ..., \boldsymbol{r}_N\}$ randomly drawn from $\mathcal{T}$ and the remapped space $\rho(\boldsymbol{x}) = \{\phi(\boldsymbol{x}, \boldsymbol{r}_1, \chi_1), \phi(\boldsymbol{x}, \boldsymbol{r}_2, \chi_2), ..., \phi(\boldsymbol{x}, \boldsymbol{r}_N, \chi_N)\}$. Let $\tilde{\phi}$ be*

$$\tilde{\phi}(\boldsymbol{x}, \boldsymbol{r}) = \frac{1}{L}(\delta(\boldsymbol{r}, \boldsymbol{r}_1)\phi(\boldsymbol{x}, \boldsymbol{r}_1, \chi_1) + \delta(\boldsymbol{r}, \boldsymbol{r}_2)\phi(\boldsymbol{x}, \boldsymbol{r}_2, \chi_2) + ... + \delta(\boldsymbol{r}, \boldsymbol{r}_N)\phi(\boldsymbol{x}, \boldsymbol{r}_N, \chi_N)) \quad (2.12)$$

where $\delta(\boldsymbol{r}, \boldsymbol{r}_j)$ is the Kronecker delta. If $\tilde{\phi}$ is an (ϵ, γ)-good similarity function for a generic sample of size N of the learning problem P then with probability $1 - \epsilon$ the mapping $\rho : \mathcal{X} \to \mathbb{R}^N$ with $N = (8 \cdot \log(1/\delta))/\gamma^2$ has the property that the induced distribution $\rho(\boldsymbol{P})$ in $\mathbb{R}^N$ has a separator of error at most $\epsilon + \delta$ at margin $\gamma/2$.

Proof. Following proof of Theorem 3 [27], $\tilde{\psi} : \mathcal{X} \to \mathbb{R}^N$ defined as $\tilde{\psi}(\boldsymbol{x}) = \frac{\tilde{\rho}(\boldsymbol{x})}{\sqrt{N}}$ $\tilde{\rho}(\boldsymbol{x}) = \{\tilde{\phi}(\boldsymbol{x}, \boldsymbol{r}_1, \chi_1), \tilde{\phi}(\boldsymbol{x}, \boldsymbol{r}_2, \chi_2), ..., \tilde{\phi}(\boldsymbol{x}, \boldsymbol{r}_N, \chi_N)\}$ with probability $1 - \delta$, the induced distribution $\tilde{\psi}(\boldsymbol{P}) \in \mathbb{R}^N$ would have a separator of error at most $\epsilon + \delta$ and margin at least $\gamma/2$. Let $\tilde{\boldsymbol{\beta}}$ be the vector corresponding to such separator, and convert it into $\hat{\boldsymbol{\beta}} \in \mathbb{R}^{N \times N}$ by replacing each coordinate $\tilde{\beta}_i$ with the N values $(\frac{1}{N}\tilde{\beta}_i, \frac{1}{N}\tilde{\beta}_i, ..., \frac{1}{N}\tilde{\beta}_i)$. Given that $\|\hat{\boldsymbol{\beta}}\| = \|\tilde{\boldsymbol{\beta}}\|$ then the margin in the space $\mathbb{R}^{N \times N}$ is $\gamma/2$. By construction $\rho(\boldsymbol{x}) = \tilde{\rho}(\boldsymbol{x})$, then the same linear separator holds for the learning problem $\boldsymbol{P}$. □

Theorem 2.1 assumes -without any loss in generality- $\boldsymbol{r} \in \mathcal{T}$. Sec. 2.3.2 will show that when sampling $\boldsymbol{r}$ in $\mathbb{R}^D$ one actually needs to suitably increase the number of landmarks N.

Interestingly, this theorem shows that the ELM model also can exploit the notion of "good" similarity function. In the hypothesis space (2.10), the assumption is that such

similarity function is "good" all over the input domain. On the other hand, the ELM model exploits a similarity function that becomes "good" by adapting its configuration to the landmark at-hand.

Finally, it is worth noting that the ELM model can also adopt mapping layers where $\Phi_1 \neq \Phi_2 \neq .... \neq \Phi_N$, i.e., where each neuron exploits a specific activation/similarity function. Theorem 2.1 actually can be easily extended to this configuration.

2.3.2 Landmarks Sampling

The theory of learning with similarity functions assumes that landmarks $\boldsymbol{r}_j$ should lie inside the input domain $\mathcal{X} \subseteq \mathbb{R}^D$. Accordingly, one needs $N = (8 \cdot \log(1/\delta))/\gamma^2$ landmarks to obtain a low-error ($\leq \epsilon + \epsilon^*$) large-margin ($\geq \gamma/2$) separator in the remapped space.

The ELM strategy, conversely, extends the admissible domain for the landmarks to $\mathbb{R}^D$. Formally, the probability of getting a landmark belonging to $\mathcal{X}$ by randomly sampling $\mathbb{R}^D$ can be modelled as a Bernoulli distribution:

$$P_{\boldsymbol{r}\in\mathcal{X}} = P(\boldsymbol{r} \in \mathcal{X} | \boldsymbol{r} \in \mathbb{R}^D) \leq 1 \tag{2.13}$$

This in turn means that given $\hat{N}$ random patterns, the probability of having at least N admissible landmarks in the sense of the theory of similarity functions can be expressed as the cumulative of a binomial distribution:

$$F(N, \hat{N}, P_{\boldsymbol{r}\in\mathcal{X}}) = \sum_{i=0}^{N} \binom{N}{i} (P_{\boldsymbol{r}\in\mathcal{X}})^i (1 - P_{\boldsymbol{r}\in\mathcal{X}})^{\hat{N}-i} \tag{2.14}$$

then, with probability $\zeta = 1 - F(N, \hat{N}, \alpha)$ at least N samples belong to $\mathcal{X}$.

In practice, this means that by applying the ELM strategy one needs at least $\hat{N} > N$ landmarks to obtain a low-error ($\leq \epsilon + \epsilon^*$) large-margin ($\geq \gamma/2$) separator in the remapped space. Obviously, the exact value of $\hat{N}$ is unknown as, in general, $P_{\boldsymbol{r}\in\mathcal{X}}$ is unknown. It is important to note that this is a worst case analysis, because the assumption here is that patterns that do not belong to input domain $\mathcal{X}$ are not involved at all in the learning phase.

2.3.3 Comparison Summary

Subsections 2.3.1 and 2.3.2 state that the two learning paradigms not only share some similarities in the shape of the eventual predictors, but importantly can be reinterpreted under a unique theoretical framework with some important differences in the rationale behind the models.

The ELM model extends the sampling domain of acceptable landmarks from $\mathcal{T}$ to $\mathcal{X}$ to bypass any constraint on the size, N, of the mapping layer. This policy may prove especially useful in the presence of limited datasets, i.e., when $\mathcal{T}$ might not properly cover the (unknown) distribution $\boldsymbol{P}$ that characterize the learning problem at-hand without incur in overfitting. Conversely, the availability of large datasets possibly undermines the benefits of such a speculative strategy.

The choice of allowing multiple configurations of the similarity notion can in principle limit the computational complexity of the training process. When using a parameterized similarity function, model selection is the only effective method to find a suitable setting of the hyper-parameter among a variety of candidates. This clearly would bring about a computational overhead. The ELM policy in practice bypasses this issue by using randomization in the setup of hyper-parameters.

In summary, the speculative approach adopted by the ELM model in the setup of $\{\boldsymbol{r}_j, \chi_j\}$ aims at balancing sample coverage and computational efficiency. On the other hand, Section 2.4 shows that random settings can lead to unsuitable configurations. One faces the risk of collecting a large number of ineffective mapping units, that is, pairs $\{\boldsymbol{r}_j, \chi_j\}$ that do not support a proper remapping of data according to the embedded similarity notion.

2.4 Geometrical Analysis

Computational complexity of SLFN is directly related to the number of neurons N. In fact, N affects both the training time and computational cost of the predictor. Random based procedures clearly privilege a fast training stage at the expense of a lower accuracy of the eventual predictor, when N is fixed. Computationally demanding pruning or selection strategies [40–43] for the setup of the hidden layer are the only available option when the goal is a better trade-off between the number of neurons and generalization capabilities of predictors derived by random based models. On the other hand, a careful management of the sampling strategy can largely improve the performance of the predictor with a modest number of neurons, because it can help in avoiding pathological configurations. As an example, neurons that provide always the same output

TABLE 2.1: Commonly used activation functions

Activation Function	$\phi(I)$
Triangular Basis Function (TBF)	$\begin{cases} 1 & I = 0 \\ -I & 0 < I < 1 \\ 0 & I \leq 1 \end{cases}$
Radial Basis Function (RBF)	$= e^{-I^2}$
MultiQuadric (MQ)	$= \sqrt{I^2}$
Inverse MultiQuadric (IMQ)	$= \frac{1}{\sqrt{1+I^2}}$
Sigmoid (SGM)	$= \frac{1}{1+e^{-I}}$
Threshold (THR)	$\begin{cases} 1 & I > 0 \\ -1 & I \leq 0 \end{cases}$

independently by the input are useless. In the following two subsections, a series of considerations about the role played by the hidden layer parameters is presented for different activation functions. Firstly, the role played by different parameters in common activation functions is discussed in subsection 2.4.1. Secondly the peculiarities of threshold function are targeted in subsection 2.4.2.

2.4.1 Generic Activation Function

Let g be a bounded, monotonically decreasing transfer function; then, one has:

$$\phi(\boldsymbol{x}, \boldsymbol{r}, \chi) = g(I(M(\boldsymbol{x}, \boldsymbol{r}), \chi)) \tag{2.15}$$

Equation (2.15) shows that the activation/similarity function applies the transfer function, g, on an input $I(M(\boldsymbol{x}, \boldsymbol{r}), \chi)$. In this notation, M denotes the metric that process the datum $\boldsymbol{x}$ and the landmark $\boldsymbol{r}$, while the complete input I derives by the action of the shape parameter χ on the metric. Table 2.1 provides some examples of activation functions that indeed satisfy the ELM universal-approximation capability theorems [13]. All the functions presented in table 2.1 can be converted in monotonically decresasing function with a sign inversion where needed.

The following property points out a basic attribute of the class of monotonic similarity functions (2.15):

Property 1. The relative placement in the similarity domain of any two samples $\boldsymbol{x}_1, \boldsymbol{x}_2$ with respect to $\boldsymbol{r}$ is independent of χ and g.

This means that, given a landmark $\boldsymbol{r}$ and a pair of samples $\{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ such that $M(\boldsymbol{x}_1, \boldsymbol{r}) < M(\boldsymbol{x}_2, \boldsymbol{r})$, the following property holds for any positive value of χ :

$$g(I(M(\boldsymbol{x}_1, \boldsymbol{r}), \chi)) \geq g(I(M(\boldsymbol{x}_2, \boldsymbol{r}), \chi)).$$

To understand the consequences of Property 1, let ν_C^- be the distance M between $\boldsymbol{r}$ and the closest sample of class '-1', and ν_O^- be the distance M between and the outermost sample of class '-1'. Likewise, let ν_C^+ and ν_O^+ be the corresponding quantities for class '+1'. Then, a prerequisite to avoid class overlapping in the similarity domain is

$$\nu_C^- < \nu_O^- < \nu_C^+ < \nu_O^+ \tag{2.16}$$

or, equivalently,

$$\nu_C^+ < \nu_O^+ < \nu_C^- < \nu_O^- \tag{2.17}$$

In practice, the prerequisite for obtaining the configuration of fully separated classes in the similarity domain is to use a landmark, $\boldsymbol{r}$, that satisfies the above conditions. In fact, g and χ cannot affect the relative order of the samples in the similarity domain.

A second attribute of the class of similarity functions (2.15) is formalized by the following proposition

Property 2. The similarity between any sample $\boldsymbol{x}$ and $\boldsymbol{r}$ is a function of both f and χ.

Let TBF be the activation/similarity function adopted with $I(M(\boldsymbol{x}, \boldsymbol{r}), \chi) = \chi \|\boldsymbol{x} - \boldsymbol{r}\|^2$. The Euclidean distance assesses the dissimilarity between an input sample, $\boldsymbol{x}$, and the reference sample (i.e., the landmark) $\boldsymbol{r}$. The specific shape of the transfer function, TBF, converts dissimilarities into similarities. Thus, when $\boldsymbol{x} = \boldsymbol{r}$, one has maximum similarity. Conversely, when $\boldsymbol{x} \neq \boldsymbol{r}$, the degree of similarity may not decrease linearly as the Euclidean distance increases.

Figure 2.1 illustrates the remapping action performed by this function on a simple case study. Figure 2.1(a) shows the samples lying on a two-dimensional space, where classes are represented by triangular patterns ('-1') and square patterns ('+1'). The diamond symbol marks the position of the landmark, $\boldsymbol{r}$. Figure 2.1(b) presents the remapping action performed by TBF, which ascribes null similarity to any sample that lies at a distance greater than $1/\chi$ from the landmark. The figure also marks the four basic quantities defined above, that is, $\nu_C^+, \nu_O^+, \nu_C^-$ and ν_O^-.

The figure presents the most profitable configuration: the landmark has been selected so that an appreciable margin separates the Z^+ samples of class '+1' from the Z^- samples of class '-1'. Moreover, the transfer function g preserves (or even emphasizes) such

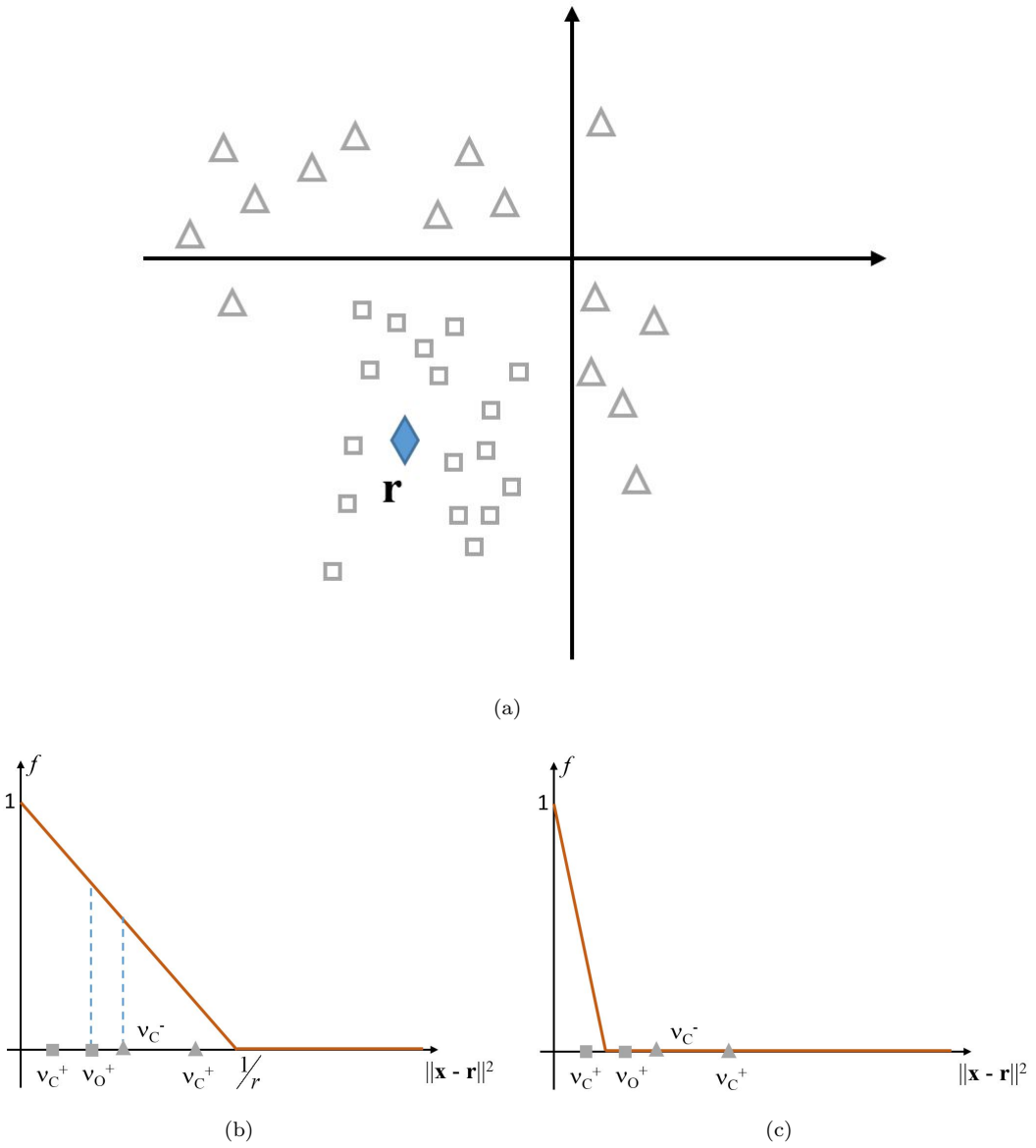


FIGURE 2.1: An example of data remapping with TBF: (a) original input space; (b) data remapping with a suitable value of r; (c) data remapping with an inappropriate value of r.

margin. This in turn means that such configuration would lead to a strongly $(0, \gamma)$-good similarity function.

Figure 2.1(c) illustrates a counterexample. If χ takes on a value such that $1/\chi < \nu_O^+$, a portion of the samples of class '+1' becomes indistinguishable - in the similarity domain - from the samples of class '-1'. This situation may occur even in the presence of a wide margin between the two classes with respect to $\boldsymbol{r}$. The worst case occurs when one sets $1/\chi < \nu_C^+$: all samples become identical after remapping, since the similarity between $\boldsymbol{r}$ and any sample always nullifies.

The above discussion applies in general to any activation function described by equation (2.15). The pair $\{g, \chi\}$ sets a specific similarity based on the mapping $M(\boldsymbol{x}, \boldsymbol{r})$ whose behavior and properties stem from the combined contributions of two main factors:

- First, thanks to the monotonic nature of g (as per (2.15)) and the radial mapping, the ordering is not altered by the specific similarity notion. In terms of the (ϵ, γ) formalism, the landmark $\boldsymbol{r}$ establishes the level of overlap between the two classes and therefore sets the smallest attainable value of ϵ.
- Secondly, the specific resulting similarity metric defined by $\{g, \chi\}$ determines the gap between a sample and its neighbors

2.4.2 Threshold Function and Scalar Product

Between the existing activation function, the threshold is probably the most convenient if one targets a digital implementation. On the other hand, this function cannot be selected when the training process involves the use of the BP algorithm because it is non differentiable. Many approximated approaches have been proposed in the literature [44, 45] but these approximations induce performance's deterioration. Interestingly, Huang et al. [46] proved that the ELM theory also holds for hard-limiter activation functions extending as a consequence, universal approximation capabilities to directly trained threshold networks.

In general, the threshold function can be considered as a degenerate case of the sigmoid function where the slope of the non saturating portion is infinite. In this section details about the scalar product geometrical distribution and its interaction with sigmoid function are analyzed. As a major result, it is shown that sigmoid based networks contain a considerable percentage of neurons that acts as threshold unit.

The scalar product activation can be conveniently rewritten as:

$$I(M(\boldsymbol{x}, \boldsymbol{r}), \chi) = \boldsymbol{x}^t \boldsymbol{r} + \chi = \boldsymbol{x}^t \boldsymbol{r} + \boldsymbol{l}^t \boldsymbol{r} = \|\boldsymbol{r}\|(\boldsymbol{x}^t \hat{\boldsymbol{r}} + \boldsymbol{l}^t \hat{\boldsymbol{r}}) = \|\boldsymbol{r}\|(\boldsymbol{x}^t \hat{\boldsymbol{r}} - r) \tag{2.18}$$

where $\hat{\boldsymbol{r}}$ is a unit vector and $r \in \mathbb{R}$ is the projection of a generic point $\boldsymbol{l}$ onto $\boldsymbol{r}$. As a result, $I(M(\boldsymbol{x}, \boldsymbol{r}), \chi)$ can be reinterpreted as $I(\boldsymbol{x}, \boldsymbol{r}, r)$. If $\boldsymbol{x} \in [0, 1]^D$, without any loss of generality, the scalar projection of $\boldsymbol{x}$ on $\hat{\boldsymbol{\omega}}$ lies in an interval that depends on the dimensionality D of the input space; i.e., $\boldsymbol{x}^t \hat{\boldsymbol{r}} \in [-\sqrt{D}, \sqrt{D}]$.

According to equation (2.18), in a sigmoid function, $SGM(I) \in [0, 1]$, the high-gradient portion of the curve approximately lies in the range

$$\left[r - \frac{5}{\|r\|}, r + \frac{5}{\|r\|}\right] \tag{2.19}$$

Since it would not be convenient to center the sigmoid out of this interval, one eventually sets, $r \in [-\sqrt{D}, \sqrt{D}]$.

Equation (2.19) proves that $\|\boldsymbol{r}\|$, plays a role in shaping SGM(I). To evaluate the effect of $\|\boldsymbol{r}\|$, let ϵ be the ratio between the non-saturating portion of the sigmoid function (2.19) and the interval in which the projected patterns lie (i.e., $2\sqrt{D}$)

$$\epsilon \leq \frac{10}{2\sqrt{D}\|\boldsymbol{r}\|} = \frac{5}{\sqrt{D}\|\boldsymbol{r}\|} \tag{2.20}$$

The upper bound (2.20) to ϵ holds as the quantity (2.19) may partially lie outside the range $[-\sqrt{D}, +\sqrt{D}]$ (e.g., when $r = \pm\sqrt{D}$). Obviously, $\epsilon = 0$ means that the sigmoid function degenerates into a hard-limiter function (i.e., $\|\boldsymbol{r}\| \to \infty$).

In the ELM model, $\boldsymbol{r}$ is usually drawn from a uniform distribution; i.e., $P(r_d) = U(-1, 1)$. By suitably extending the Central Limit Theorem [47], one can approximate the distribution of $\|\boldsymbol{r}\|$ as a normal distribution in the range $[0, \sqrt{D}]$ whose expectation is $\mu_{\|\boldsymbol{r}\|} = \sqrt{Var(r_d)}\sqrt{D} = 0.57\sqrt{D}$. The expression (2.20) shows that in the basic ELM training protocol 50% of the hidden activation functions have $\epsilon \leq 0.3$ as long as $D > 30$. When $D = 100$ one has $\epsilon < 0.09$. As a result, an ELM with sigmoid functions always embeds a subset of neurons that actually involve threshold mechanisms, even if $\|\boldsymbol{r}\| \in [0, \sqrt{D}]$. The incidence of the latter neurons increases as the dimensionality D of the input space grows. So one might set up an ELM that only includes hard-limiter functions, with the purpose of obtaining a predictor explicitly designed for digital low cost implementations. Such a strategy might bring about a reduced generalization ability, as compared with a configuration that relies on sigmoid functions. The above discussion suggests that this gap may not be substantial.

Finally, it is important to note the different role played by the parameter r in the threshold and sigmoid. Consider the example shown in figure 2.2(a) where the action performed by a single remapping unit of the form $g(\hat{\boldsymbol{r}} \cdot \boldsymbol{x} + r)$ is shown. In the example, the samples lie on a two-dimensional space, where classes are represented by diamond patterns ('-1') and cross patterns ('+1'). The space is divided in two parts by a hyper-plane identified by the couple $\{\hat{\boldsymbol{r}}, r\}$ that perfectly separates the two classes. Figure 2.2(b) represent the same example but with a different value for the bias r for the hyper-plane; in this case all the samples would lie on the same side of the linear separator. Figure 2.2(c) and 2.2(d) refer, respectively, to the remapping action performed by the hyper-plane presented in figures 2.2(a) and 2.2(b). In both the figures on the y-axes the remapping obtained

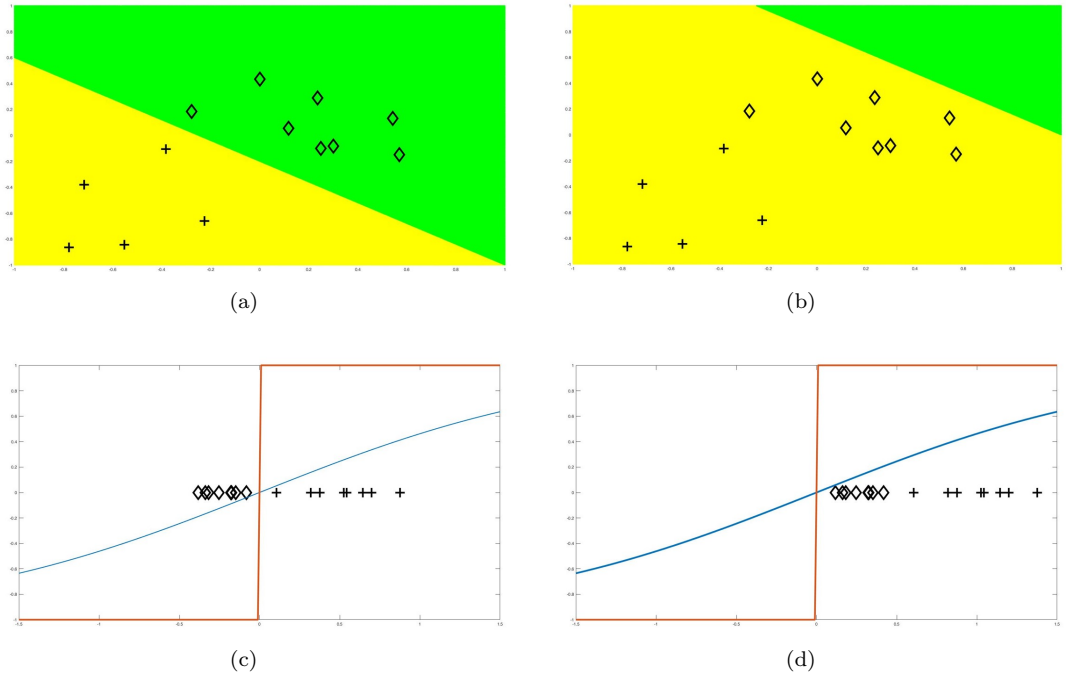


FIGURE 2.2: An example of data in 2D input space: (a) linear separator with error $\epsilon = 0$; (b) same linear separator but different bias r; (c) activation obtained by using linear separator (a) and sign activation (orange line) or rescaled sigmoid (blue line) (d) same format of figure (c) but linear separator of figure (b)

by the sigmoid function (orange line) and the sign function (blue line) are presented. The remapped data are linearly separable with error 0 for both the non linearities in figure 2.2(c). Instead, in figure 2.2(d), the remapped space obtained using the SGM function maintains the linear separability property with null error, while the projection obtained using threshold function are non-separable due to the configuration of the bias r in the hyper-plane of figure 2.2(b).

This simple example enlightened the different implication of the parameter r with different non linearities and stress the fact that a random selection of this parameter can easily lead to useless remapping unit (i.e. neurons) also when the parameter $\hat{\boldsymbol{r}}$ are optimal.

2.5 Proposed Algorithms for Efficient Mapping

As a major outcome, this chapter provides a set of algorithms for an efficient setup of the hidden layer parameters. All the proposed solutions share the common goal of yield to predictors that can efficiently balance the trade-off between computational cost of the eventual predictor and generalization ability without a significant increment of the training phase' cost. To achieve this, the proposed strategies addressed this problem by

working on the parameter χ; this avoids any re-computation of the vector operations relative to $M(\boldsymbol{r}, \boldsymbol{x})$.

2.5.1 Resampling Shape Factor Algorithm

Section 2.4 clearly depicted the different role played by $\{\boldsymbol{r}, \chi\}$ when combined with different kind of non linearities. In particular, subsection 2.4.1 exacerbate the concept that the selection of a non-consistent parameter χ could lead to pathological situation. As an example, a mapping unit that collapses all the patterns $\boldsymbol{x} \in \mathcal{X}$ in one single point $p \in \mathbb{R}$ is not useful in terms of learning. The ultimate goal of the mapping layer is to project the input samples in a new space in which positive and negative patterns are separable. In a simple one-dimensional mapping space, if μ_+ denotes the barycenter of the positive patterns on $\mathbb{R}$, and μ_- denotes the corresponding barycenter of the negative patterns, the following condition should hold:

$$\mu_+^{(j)} \neq \mu_-^{(j)}, \quad j = 1, ..., N \tag{2.21}$$

where N is the number of mapping units (i.e., the number of landmarks in the mapping layer). Ideally, the best mapping unit clearly guarantees a large margin between μ^+ and μ^- with a small intra-class variance.

In general, by randomly selecting the values to be assigned to χ_j one might end up into flawed configurations, irrespective of the specific choice of the similarity/activation function. In the ELM model, this ultimately means that a subset of neurons may be almost useless in terms of learning. Thus one may not be able to address effectively the trade-off between generalization performance and the computational complexity of the eventual predictor, which becomes critical when targeting the implementation of the classification system on electronic devices.

It is convenient to compare the conventional (blind) strategy applied by ELM for the setup of χ with a strategy that ensures a compliance with constraint (2.21). The goal of such a novel strategy is to check the configuration of parameters, χ, that is assigned to a mapping unit j; a configuration is considered flawed when:

$$|\mu_+^{(j)} - \mu_-^{(j)}| < \tau \tag{2.22}$$

where τ is a threshold value setting the tolerance admitted in constraint violation. The algorithm only validates the setting of χ_j, since both the activation/similarity function, ϕ, and the landmark $\boldsymbol{r}_j$ assigned to the j-th unit do not vary.

Algorithm 3 outlines the associate procedure, which actually rewrite Step 0 (*Initialize*) in Algorithm 1. The updated *Initialize* step receives a labeled training set $\mathcal{T} = \{(\boldsymbol{x}, y)_i; i = 1, .., Z\}$, an activation/similarity function ϕ, an admissible range of values for χ, and a target value, N, for the dimensionality of the mapping layer.

For each mapping unit, the algorithm first generates the corresponding landmark by using the conventional ELM strategy. Then, the routine for setting χ_j proceeds in two steps: 1) a random value is drawn within the input range, and 2) the value is validated. The routine stops when the step 2) completes successfully. Eventually, the Algorithm yields the values of the free parameter, χ_j, for each j-th mapping unit (neuron). Different criteria may apply to assess the effectiveness of a mapping unit at separating positive samples from negative ones. Deadlock are avoided simply dividing by two the value of the threshold τ each time that a configuration is rejected.

In Algorithm 3, the two sets $\mathcal{X}_+$ and $\mathcal{X}_-$ are eventually characterized by the average values, but other quantities such as median or p-order percentile can also be used. Furthermore, the algorithm does not consider the interaction between different remapping units and does not consider the inter and intra class variance. The logic behind these choices is to introduce a negligible overload in the computational cost of the training phase. It still appears less efficient than the original ELM procedure in terms of computational complexity, since the number of attempts required to find a valid setting, χ, for each mapping unit is not predictable. On the other hand, the strategy in Algorithm 3 can offer significant advantages in terms of trade-off between generalization performance and size of the mapping layer.

2.5.1.1 Experimental Results

The experimental section aims at evaluating the ability of Algorithm 3 to improve the overall performance of the ELM model in terms of classification accuracy. That is, the goal is to verify that by applying Algorithm 3 one can attain a mapping layer that better supports the search for a consistent (ϵ,γ)-good similarity function. To the purpose of robustly assessing such aspect, five different benchmarks [48], previously introduced in related literature, have been involved in the experimental evaluation: Ionosphere, Glass Identification, Statlog Landsat Satellite, Covertype and CodRNA. Each experimental session has been designed to provide a fair comparison between the generalization performances of two ELM models: the one that applies the conventional strategy in the setup of free parameters and the one that exploits Algorithm 3. Therefore, in each experiment, instead of employing standard model selection techniques [12], the two implementations of ELM have been compared by defining a common configuration for

Algorithm 3 Enhanced training strategy using random search

Input

- a labeled training set $\mathcal{T} = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z\}$
- number of neurons N
- range of admissible values for $\chi : [\chi_{inf}, \chi_{sup}]$
- threshold value τ

0. **Initialize**

a. generate the set of random landmarks $\boldsymbol{r}_j, j = 1..., N$

for $j = 1$ to N **do**
  for $d = 1$ to D **do**
    $\boldsymbol{r}_{j,d} = rand(-1; 1)$
  end for
end for

b. for each neuron, set χ_j by applying the following routine:

$\mathcal{X}_+ = \{\}$, $\mathcal{X}_- = \{\}$

$ok = 0$

while $ok == 0$ **do**
  $\chi_j = rand(\chi_{inf}, \chi_{sup})$
  for $i = 1$ to Z **do**
    $a_i = \phi(\boldsymbol{x}_i, \boldsymbol{r}_j, \chi_j)$
    if $y_i == 1$ **then**
      $a_i \rightarrow \mathcal{X}_+$
    else
      $a_i \rightarrow \mathcal{X}_-$
    end if
  end for
  $\mu_+ = mean(\mathcal{X}_+)$
  $\mu_- = mean(\mathcal{X}_-)$
  if $|\mu_+ - \mu_-| \geq \tau$ **then**
    $ok = 1$
  else
    $\tau = \tau/2$
  end if
end while

1. **Mapping**

remap all the patterns $\boldsymbol{x} \in \mathcal{T}$ by using the following mapping function

$$\boldsymbol{\psi}(\boldsymbol{x}) = \{\phi(\boldsymbol{x}, \boldsymbol{r}_1, \chi_1), ..., \phi(\boldsymbol{x}, \boldsymbol{r}_N, \chi_N)\}$$

2. **Learning**

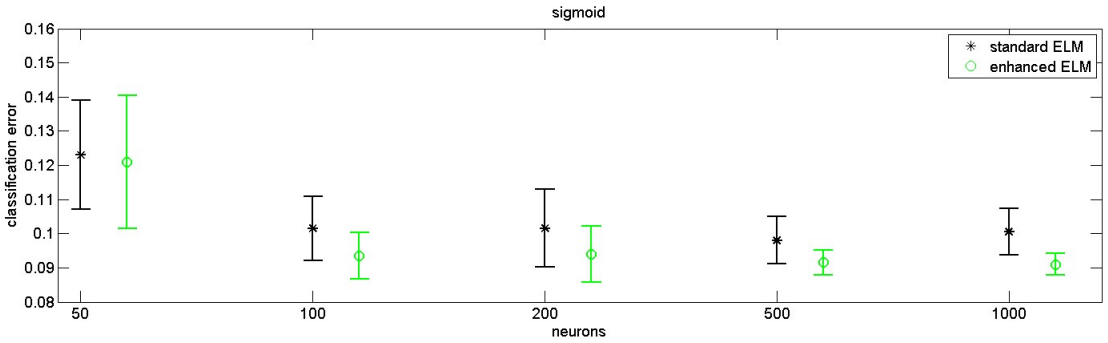
train a linear predictor in the space $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^N$

both the range of admissible λs (i.e., the regularization parameter), and the dimensionality, N, of the remapped space (i.e., the number of landmarks/neurons). The configurations are: $\lambda = \{10^{-6},\ 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$, $N = \{50, 100, 200, 500, 1000\}$. All the simulation in the present chapter were performed using Matlab software.

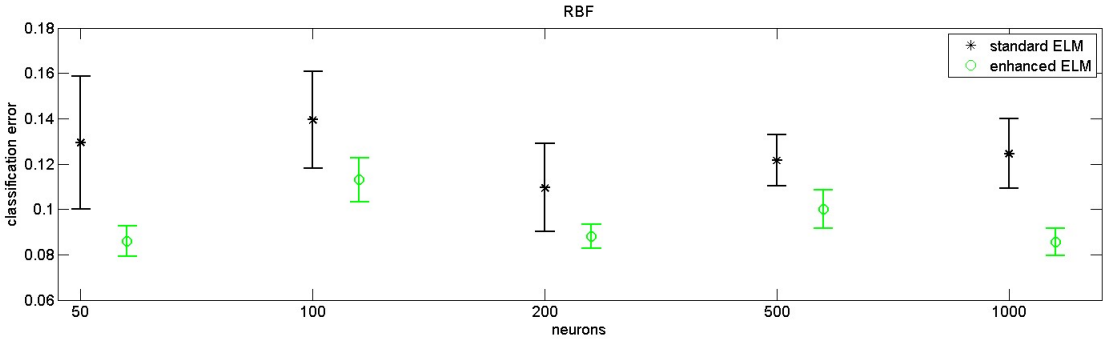
Ionosphere dataset The Ionosphere dataset includes a total of 351 patterns, which lie in a 34-dimensional space; the original dataset is quite unbalanced, as one of the two classes only provided 126 patterns out of 351. In the present experimental design, both the training set and the test set included 50 patterns per class; all the 34 features are renormalized in the interval $[-1, 1]$. Three different activation/similarity functions have been involved in the session: sigmoid, RBF, and multiquadric functions. Thus, the free parameters are the bias, the spread factor, and the spread factor, respectively. The enhanced ELM implementation exploited Algorithm 3 with threshold $\tau = 0.5$. Hence, a mapping unit is considered effective only when the gap between the barycenter μ_+ of the remapped positive samples and the barycenter μ_- of the remapped negative samples is larger than 0.5; it is worth to note that the remapped patterns lie in a space with range $[-1, 1]$.

Figure 2.3 provides the outcomes of the three experiments. Figure 2.3(a) refers to the experiment in which the mapping units implement the sigmoid function; the x axis gives the number of neurons, N, while the y axis gives the classification error (expressed as percentage over the size of the test set). The graph compares the performance of the standard ELM (asterisk as marker) with the performance of the enhanced ELM (circle as marker). For each N, the performance of a predictor is assessed by the configuration (i.e., λ) that leads to the best average classification error on the test set; the average value is computed over 50 runs, i.e., 50 different randomizations of the mapping layer. The graph also provides the confidence interval $\pm\sigma$. The same format applies to Figure 2.3(b), which refers to the experiment in which the mapping units implement the RBF, and to Figure 2.3(c), which refers to the experiment in which the mapping units implement the multiquadric function.

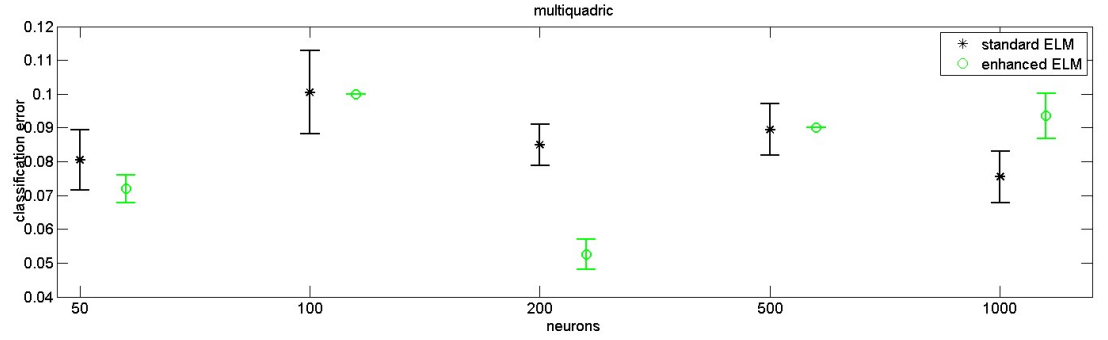
Overall, the graphs show that the enhanced ELM can, in most cases, improve over standard ELM in terms of classification performance. Indeed, the improvement varies as a function of the activation/similarity function and of the number of neurons. In this regard, Table 2.2 reports -for each activation/similarity function- the best predictor, i.e., the predictor that scored the lowest classification error. Thus, each row indicates the predictor setting (standard/enhanced), the classification error, and the number of neurons. In addition, the last row of the table reports on the performance scored by



(a)



(b)



(c)

FIGURE 2.3: Results of the experiments involving the Ionosphere dataset: a) sigmoid; b) RBF; c) multiquadric

TABLE 2.2: Comparison between ELM and SVM for the Ionosphere dataset.

Function	Best predictor	Classification Error	Configuration
Sigmoid	Enhanced ELM	9.1	N = 1000
RBF	Enhanche ELM	8.6	N = 50
Multiquadratic	Enhanche ELM	5.2	N = 200
SVM		7.0	(C,σ) = (10^{-1},1)

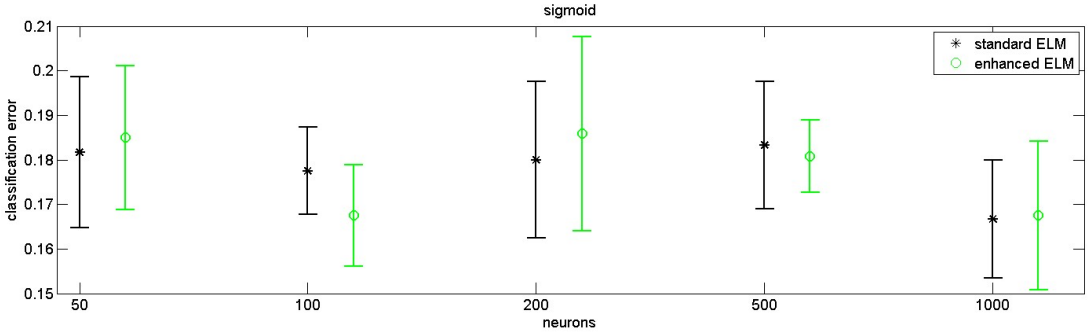
a Support Vector Machine (SVM) [12] on the same problem. The results refer to an implementation based on the RBF kernel: the table gives the classification error attained after model selection [49, 50] along with the corresponding selected configuration (C, σ).

Glass dataset The Glass Identification dataset includes 214 samples that lie in a 9-dimensional space. The benchmark involves a multi-class problem, as six different classes are represented in the dataset; the experiments presented here, though, only address a binary classification problem, namely, class 1 versus class 2. In the proposed experimental design, both the training set and the test set include 30 patterns per class randomly extracted from the original dataset. All the 9 features were renormalized in the interval $[-1, 1]$. As above, three different activation/similarity functions have been involved in the session: sigmoid function, RBF, and multiquadric function. The enhanced ELM implementation exploited Algorithm 3 with threshold $\tau = 0.5$.

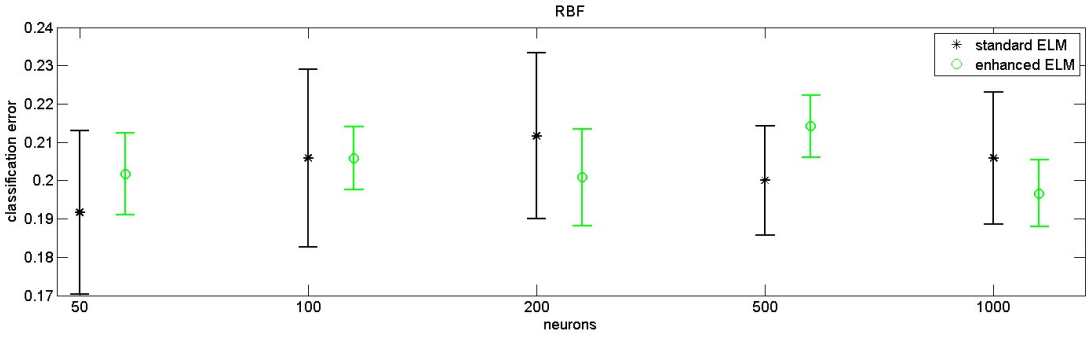
Figure 2.4 provides the outcomes of the three experiments. The format adopted for the graphs replicates the one used in Figure 2.3. Overall, the graphs show that -with this dataset- the enhanced ELM improved over standard ELM only in a few cases. Nonetheless, it is interesting to note that -when adopting the sigmoid function- the standard ELM scored its best performance with 1000 neurons (classification error of 16.7%). The corresponding predictor based on Algorithm 3 was indeed able to score the same classification error by using 100 neurons.

Table 2.3 provides the comparison between the predictors based on ELM and the predictor based on SVM. In practice, the enhanced ELM based on sigmoid function achieved the same performance of SVM (classification error of 16.7%), while the enhanced ELM based on multiquadric function scored a classification error quite close to that reference.

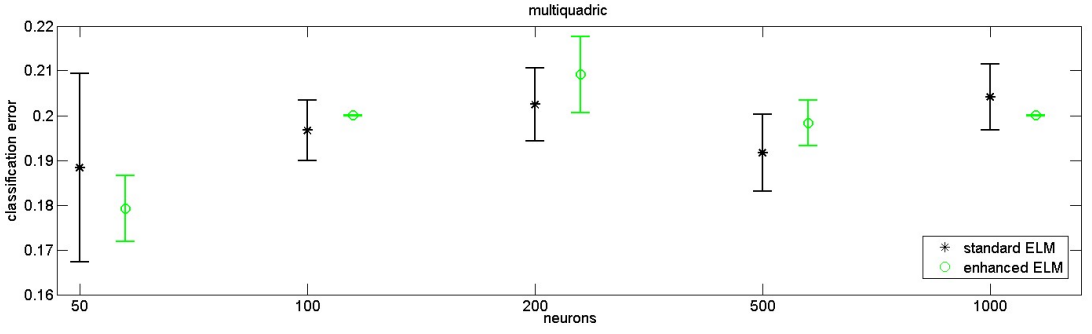
Landsat dataset The Landsat satellite dataset provides a training set including 4435 samples and a test set including 2000 samples; data are drawn from a 36-dimensional space. The original benchmark involves a multi-class problem, but the present experiments only address a binary classification problem: class 4 versus class 7. In the proposed



(a)



(b)



(c)

FIGURE 2.4: Results of the experiments involving the Glass dataset: a) sigmoid; b) RBF; c) multiquadric

TABLE 2.3: Comparison between ELM and SVM for the Glass dataset.

Function	Best predictor	Classification Error	Configuration
Sigmoid	Enhanced ELM	9.1	N = 1000
RBF	Enhanche ELM	8.6	N = 50
Multiquadratic	Enhanche ELM	5.2	N = 200
SVM		7.0	(C,σ) = (10^{-1},1)

TABLE 2.4: Comparison between ELM and SVM for the Landsat dataset.

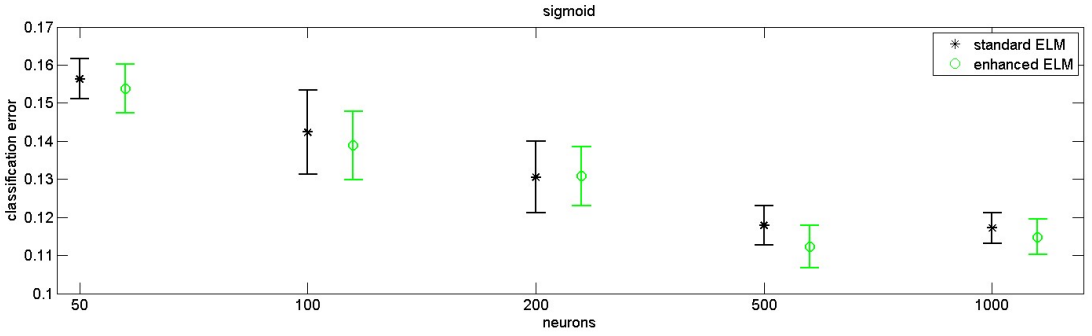
Function	Best predictor	Classification Error	Configuration
Sigmoid	Enhanced ELM	11.2	N = 500
RBF	Enhanced ELM	13.5	N = 500
Multiquadratic	Enhanced ELM	12.0	N = 500
SVM		10.3	(C,σ) = (10^{-2},1)

experimental design, the training set includes 300 patterns per class randomly extracted from the original training database; the test set includes 150 patterns per class randomly extracted from the original test database. All the 36 features have been renormalized in the interval $[-1, 1]$.

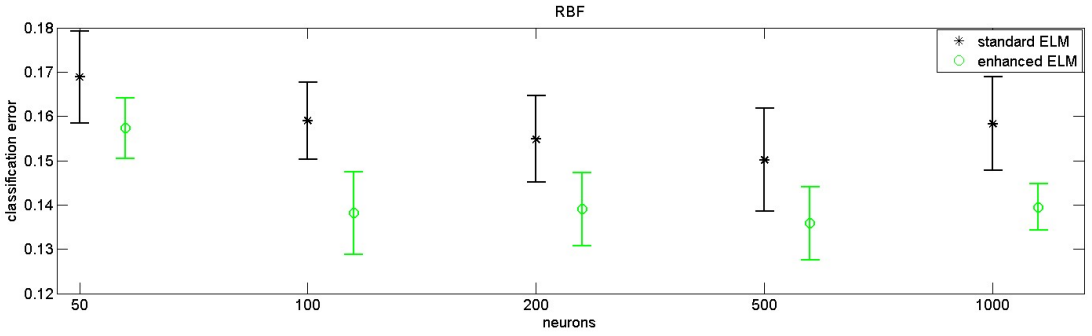
As above, three different activation/similarity functions have been involved in the session: sigmoid function, RBF, and multiquadric function. The enhanced ELM implementation exploited Algorithm 3 with threshold $\tau = 0.5$. Figure 2.5 provides the outcomes of the three experiments. The format adopted for the graphs replicates the one used in Figure 2.3. The graphs reveal that in most cases the enhanced ELM improved over standard ELM. Indeed, the gap between standard ELM and enhanced ELM is significant in particular with the RBF and the multiquadric function.

Table 2.4 provides the comparison between the predictors based on ELM and the predictor based on SVM. In this case, the enhanced ELM did not achieve the classification error scored by SVM. Nonetheless, the gap between the best result attained by the enhanced ELM (11.2%) and the classification error attained by SVM (10.3%) is not large.

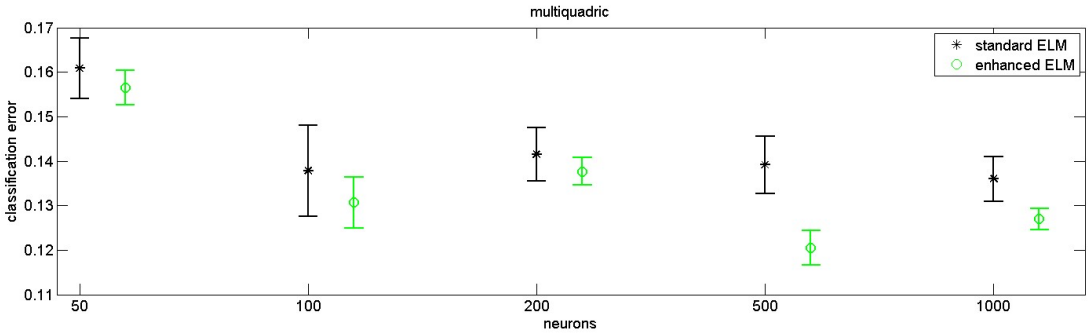
Covertype dataset The Covertype dataset provides a training set including 59,535 samples and a test set including 271,617 samples, drawn from an 8-dimensional space. In the proposed experiment, both the training set and the test set included 5000 patterns per class; all the 8 features were indeed renormalized in the interval $[-1; 1]$. As above,



(a)

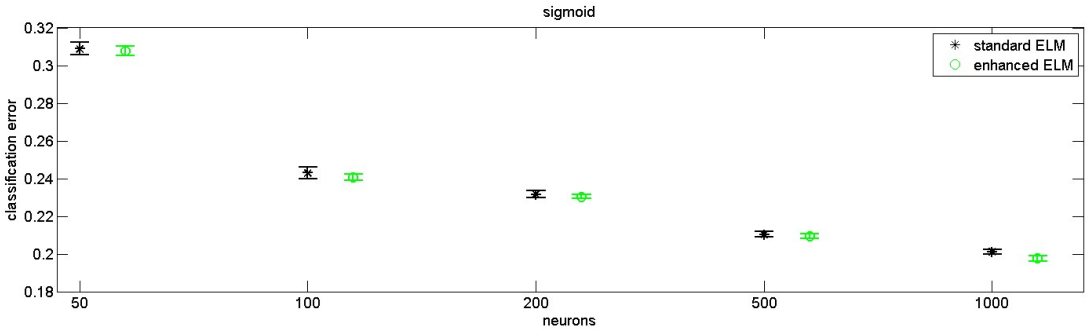


(b)

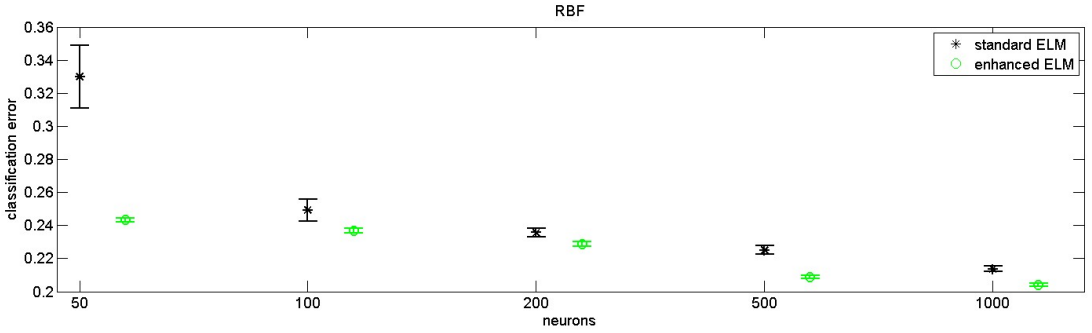


(c)

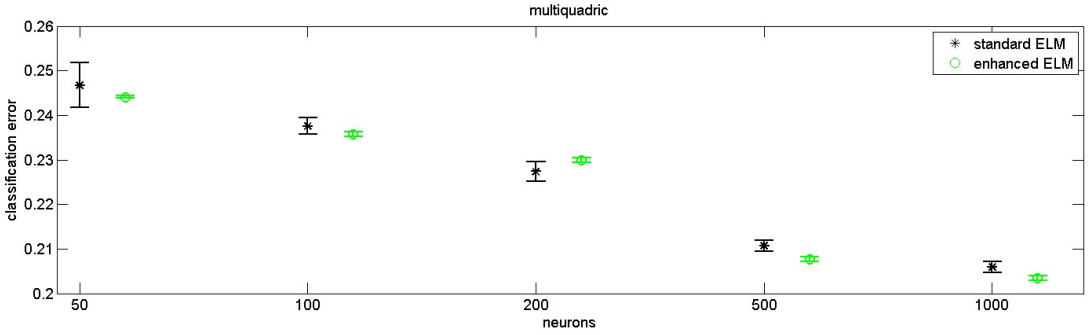
FIGURE 2.5: Results of the experiments involving the Landsat dataset: a) sigmoid; b) RBF; c) multiquadric



(a)



(b)



(c)

FIGURE 2.6: Results of the experiments involving the CovType dataset: a) sigmoid; b) RBF; c) multiquadric

three different activation/similarity functions have been involved in the session: sigmoid function, RBF, multiquadric function. The enhanced ELM implementation exploited Algorithm 3 with threshold $\tau = 0.5$. Figure 2.6 provides the outcomes of the three experiments. The format adopted for the graphs replicates the one used in Figure 2.3. The graphs reveal that the enhanced ELM improved significantly over standard ELM only with the RBF as activation/similarity function. On the other hand, the improvement is small when adopting the sigmoid function.

TABLE 2.5: Comparison between ELM and SVM for the Covertype dataset.

Function	Best predictor	Classification Error	Configuration
Sigmoid	Enhanced ELM	19.7	N = 1000
RBF	Enhanche ELM	20.4	N = 1000
Multiquadratic	Enhanche ELM	20.3	N = 1000
SVM		16.8	(C,σ) = (10^1,0.1)

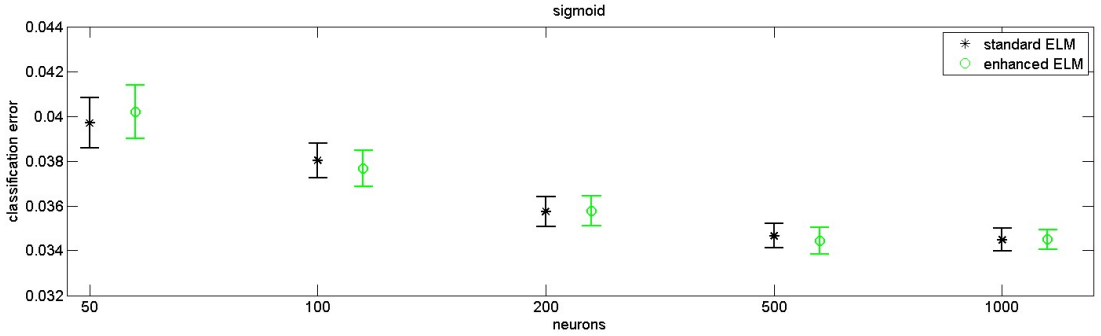
Table 2.5 provides the comparison between the predictors based on ELM and the predictor based on SVM. In this case, the performance achieved by SVM (classification error of 16.8%) is definitely better than the best performance scored by ELM.

CodRNA dataset The Cod-RNA dataset provides a training set including 59,535 samples and a test set including 271,617 samples, drawn from an 8-dimensional space. In the proposed experiment, both the training set and the test set included 5000 patterns per class; all the 8 features were indeed renormalized in the interval $[-1; 1]$. As above, three different activation/similarity functions have been involved in the session: sigmoid function, RBF, and multiquadric function. The enhanced ELM implementation exploited Algorithm 3 with threshold $\tau = 0.5$. Figure 2.7 provides the outcomes of the three experiments. The format adopted for the graphs replicates the one used in Figure 2.3. Again, the graphs show that the enhanced ELM improved over standard ELM in particular when using the RBF as activation/similarity function.

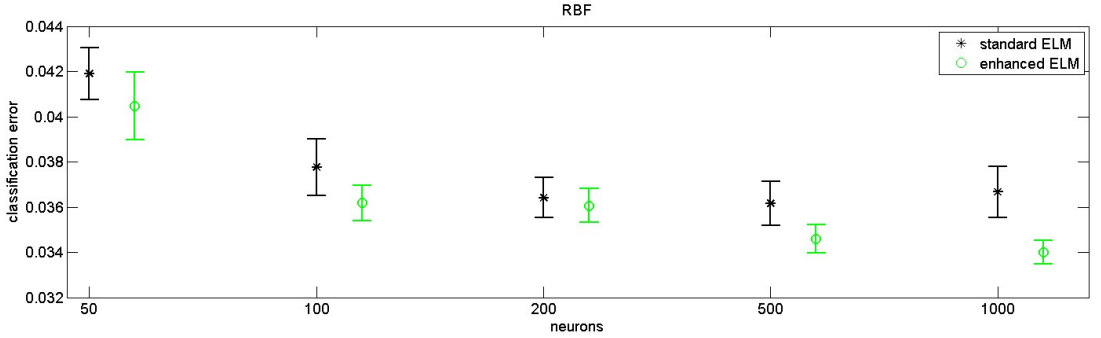
Table 2.6 provides the comparison between the predictors based on ELM and the predictor based on SVM. Numerical results reveal that the predictors based on enhanced ELM always improved over SVM. Such outcome indeed confirms that enhanced ELM may attain very interesting performance.

2.5.1.2 Concluding Remarks

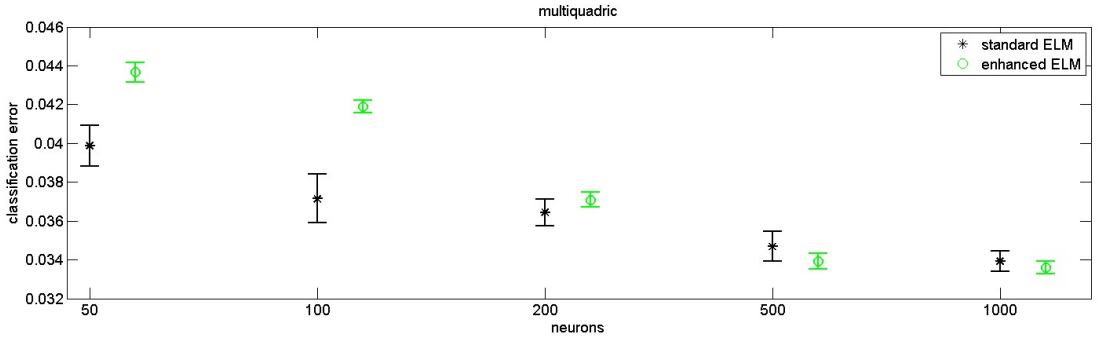
Empirical evidence supported the proposed solution, which in general allows one to obtain a more effective predictor with respect to the conventional approaches to ELM training with a minimum increment of the training phase's cost. Actually, it is worth to note that in some cases enhanced ELM showed able to reach the performances of a powerful classification system such as SVM.



(a)



(b)



(c)

FIGURE 2.7: Results of the experiments involving the CodRna dataset: a) sigmoid; b) RBF; c) multiquadric

TABLE 2.6: Comparison between ELM and SVM for the Cod-rna dataset.

Function	Best predictor	Classification Error	Configuration
Sigmoid	Enhanced ELM	3.4	N = 500
RBF	Enhanche ELM	3.3	N = 1000
Multiquadratic	Enhanche ELM	3.3	N = 1000
SVM		3.7	(C,σ) = (1,1)

2.5.2 Threshold Parameter Algorithm

A learning procedure for an ELM based on hard-limiter functions starts by setting randomly the pairs $\{\hat{\boldsymbol{r}}, r\}$, as per equation (2.18). However, the peculiarities of the specific activation function allow some improvements considering the outcome of analysis proposed in subsection 2.4.2.

The scalar value, r, in eq. (2.4.2) sets the hard-limiter threshold. Projecting any input pattern, $\boldsymbol{x} \in \mathcal{X}$, onto $\hat{\boldsymbol{r}}$ according to 2.4.2 relates $\boldsymbol{x}$ to r, i.e., to the corresponding landmark position, $\boldsymbol{l}$ as per (2.18). This approach spans a pair of subsets in the input space, according to the associate projections of each pattern with respect to $\boldsymbol{r}$. The two subsets can be formalized as

$$\Omega_{un} = \{(\boldsymbol{x}, y)_i | \boldsymbol{x}^t \hat{\boldsymbol{r}} < r\}, \Omega_{on} = \{(\boldsymbol{x}, y)_i | \boldsymbol{x}^t \hat{\boldsymbol{r}} \geq r\} \tag{2.23}$$

In practice, eq. (2.23) shows that data are projected in two sets $(\Omega_{un}, \Omega_{on})$. The most profitable, yet unlikely, situation occurs when a subset only includes the patterns belonging to one class (e.g., $y = -1$), whereas the other only covers the samples belonging to the other class (e.g., $y = +1$). In fact, an unsuitable choice of $\hat{\boldsymbol{r}}$ can make this goal unfeasible, independently of the value taken on by r: this is usually the case when $\hat{\boldsymbol{r}}$ is set randomly. However, given $\hat{\boldsymbol{r}}$, one can set the value of r so as to attain the best feasible partitioning. This is the rationale behind the training procedure that is introduced in this section to generate the pairs $\{\hat{\boldsymbol{r}}, r\}_n$ with a minimum computational overload.

First, the positions $\hat{\boldsymbol{r}}$ are set at random. Secondly, for each neuron, a pool of C candidate values for r are drawn at random. Finally, the eventual value of r_n is picked out according to some optimality criterion. Given the threshold function binary nature the maximum entropy gain [51] has been selected as optimality criterion, because the maximum entropy gain coincides with best classes separation. Accordingly, the entropy, $H(\Omega)$, of a set Ω stems from the proportion between the patterns of either classes. Algorithm 4 outlines

the complete algorithm, which yields the pairs $\{\hat{\boldsymbol{r}}, r\}_n$ to be used in the learning problem (2.11).

Since the optimality criterion is applied to a pre-selected pool of candidate values for r_n, the algorithm is not expected to find the best value for the threshold, given $\hat{\boldsymbol{r}}_n$. However, by tuning C one can balance the computational cost of the algorithm and the generalization performance of the network. As in the case of the random search, the algorithm avoids the re-computation of the scalar products that characterize the activation function, limiting in this way the overload in the training phase.

Algorithm 4 Enhanced mapping strategy for threshold activation functions using maximum entropy gain

Input

- a labeled training set $\mathcal{T} = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z\}$
- number of neurons N
- number of candidate C

0. **Initialize**
a. generate the set of random landmarks $\boldsymbol{r}_j, j = 1..., N$
b. for each neuron, set r_j by applying the following routine:

- generate a set of candidate values $\Delta = \{r_c, c = 1, ..., C\}$ with $r_c = unifrnd(-\sqrt{D}, \sqrt{D})$
- build the subset $\Omega_u(r_c)$,$\Omega_o(r_c)$ for each $r_c \in \Delta$
- set $r_{c^*} = argmin_c\{H(\Omega_u(r_c)) + H(\Omega_o(r_c))\}$

1. **Mapping**
remap all the patterns $\boldsymbol{x} \in \mathcal{T}$ by using the following mapping function

$$\boldsymbol{\psi}(\boldsymbol{x}) = \{\phi(\boldsymbol{x}, \boldsymbol{r}_1, r_{1^*}), ..., \phi(\boldsymbol{x}, \boldsymbol{r}_N, r_{N^*})\}$$

2. **Learning**
train a linear predictor in the space $\boldsymbol{\psi} : \mathcal{X} \rightarrow \mathbb{R}^N$

2.5.2.1 Experimental Results

The experimental tests address the ability of the learning procedure (Algorithm 4) at balancing accuracy and the size of the hidden layer. The generalization performance of the learning procedure has been assessed by comparing the performance of three machines: a standard ELM based on the sigmoid function (ES), a standard ELM based on the hard-limiter function (E-L), and a hardware-friendly ELM based on the hard-limiter

function (HE). The first two predictors were actually trained by using the conventional learning procedures associated to the ELM model (as per Sec. 2.1). The latter predictor, conversely, exploited a training process supported by the algorithm 4.

This session involved 15 datasets from the UCI repository [48]: Breast Cancer Wisconsin (Original, Diagnostic, and Prognostic), Blood Transfusion, Connectionist Bench, Default of Credit Card Clients, Detect Malicious Executable, Ionosphere, LSVT voice rehabilitation, MAGIC gamma telescope, Ozone level detection, Pima Indians Diabetes, Planning relax, QSAR biodegradation, and Statlog Australian credit approval. Differently from the previous experiments, only originally bi-class datasets were involved in the experimental campaign. All the data were normalized in the range $[0, 1]$. In each experiment, the predictors were compared by adopting three configurations for the number of neurons: $N = \{25, 100, 500\}$. Model selection involved the following range for the regularization parameter: $\lambda \in \{2^i, i = -10, -9, \ldots, 10\}$. In the training of HE the number of candidates for l was set to $C = 3$. Each experiment involved five runs, i.e., five random training/test pairs; 70% of patterns included in the dataset were used as a training set and the remaining patterns made the test set. The accuracy of each run was assessed by averaging over 40 extractions of the random parameters.

Table 2.7 summarizes the results of the first session of experiments, in which the learning problem involved the conventional l_2 regularization term (2.5). The first column marks the dataset. The second column gives the dimensionality, D, and the size of the dataset, respectively. The third column gives the average classification error on the test set (as percentage expressed in the range $[0, 1]$) achieved by E-S, for $N = 25$. The fourth and fifth columns give, respectively, the gain/loss in the classification error achieved by E-L and HE with respect to E-S. Thus, a negative quantity indicates that the predictor reduced the classification error with respect to ES. The same format has been applied to present the performance obtained with $N = 100$ and $N = 500$, respectively.

Numerical results clearly lead to three important outcomes. First, given a dataset, the gap between the proposed HE and ES is often smaller than the corresponding gap between E-L and E-S. Thus, HE outperforms E-L almost always. Second, the gap between HE and E-S most of the times is inferior to 4%. This in turn means that algorithm 4 is able to balance accuracy and size of the hidden layer. Third, such trend still holds when $N = 25$, i.e., when one wants to deploy a predictor that can fit low-resources devices.

The second experimental session aimed at assessing the effectiveness of the learning procedure supporting HE in limiting the occurrence of unfruitful neuron configurations. This is a major issue when targeting a proper balance between generalization performance and number of neurons. To complete such analysis, the cost function based on l_1

TABLE 2.7: Generalization performance with l_2 regularization term

Dataset	D / Size	N = 25			N = 100			N = 500		
		E-S	E-L	HE	E-S	E-L	HE	E-S	E-L	HE
BCW O	10 / 699	0.05	0.01	0.00	0.05	0.01	0.00	0.05	0.01	0.00
BWC D	32 / 569	0.02	0.05	0.01	0.01	0.02	0.00	0.01	0.01	0.00
BWC P	34 / 198	0.26	0.12	0.08	0.23	0.09	0.06	0.23	0.08	0.04
Blood	5 / 748	0.30	0.10	0.04	0.30	0.06	0.02	0.30	0.04	0.03
Conn. B.	60 / 208	0.25	0.05	0.02	0.18	0.04	0.01	0.15	0.01	0.00
Def. CC	24 / 30000	0.23	0.13	0.03	0.21	0.07	0.00	0.20	0.03	-0.01
Mal. Ex.	513 / 373	0.04	0.01	-0.02	0.02	0.00	0.00	0.02	0.00	0.00
Iono	34 / 351	0.14	0.02	0.02	0.11	-0.03	-0.03	0.09	-0.03	-0.03
LSVT	309 / 126	0.25	0.05	0.01	0.21	0.05	0.01	0.15	0.04	0.01
MAGIC	11 / 19020	0.17	0.07	0.04	0.15	0.06	0.02	0.13	0.03	0.00
Ozone	73 / 2536	0.22	0.05	0.03	0.18	0.05	0.04	0.16	0.04	0.02
Pima	8 / 768	0.13	0.06	0.02	0.13	0.03	0.01	0.13	0.02	0.00
Pl. Rel.	13 / 128	0.27	0.05	0.07	0.28	0.03	0.04	0.29	0.02	0.04
QSAR	41 / 1055	0.18	0.12	0.03	0.15	0.06	0.01	0.15	0.03	0.00
StatLog	14 / 690	0.15	0.06	0.02	0.13	0.01	0.00	0.13	0.00	0.00

regularization was employed:

$$\min_{\boldsymbol{\beta}}\{\|\boldsymbol{y} - \boldsymbol{H}\boldsymbol{\beta}\|^2 + \lambda\|\boldsymbol{\beta}\|\} \tag{2.24}$$

Actually, such learning procedure is expected to yield sparser solution by pruning useless neurons, i.e., neurons characterized by an unfruitful configuration of the parameters $\{\hat{\boldsymbol{r}}, r\}_n$. Hence, given N, it may reveal the percentage of neurons that plays an active role in the prediction, which correspond to the neurons that survived pruning.

Table 2.8 summarizes the results of this second session of experiments, which involved the same 15 datasets utilized in the first experimental session and adopted the same experimental setup. The first column gives the name of the dataset. The second, third, fourth, and fifth columns refer to the predictors with $N = 25$. The second column gives the average classification error on the test set (as percentage expressed in the range $[0, 1]$) achieved by E-S with the l_1 regularization term. The third column gives the gap in terms of classification error between E-S and HE; HE was also trained by exploiting l_1 regularization in the cost function. The classification error scored by E-S provides the reference; thus, negative quantities indicate that HE improved over E-S. The fourth column gives, as a percentage over N, the amount of neurons that survived pruning in E-S. The fifth column gives the analogous quantity for HE. The same format is used to present the performance obtained with $N = 100$. The table does not provide any comparison with E-L since Table 2.7 already proved that such predictor is less effective

Table 2.8: Generalization performance and sparsity

Dataset	N = 25				N = 100			
	E-S	HE	% Active E-S	% Active HE	E-S	E-H	% Active E-S	% Active HE
BCW O	0.05	0.00	72.6%	81.9%	0.05	0.00	39.3%	52.7%
BWC D	0.02	0.01	82.2%	88.9%	0.01	0.01	53.8%	73.0%
BWC P	0.26	0.08	75.0%	73.0%	0.23	0.06	30.4%	40.0%
Blood	0.31	0.03	26.0%	61.5%	0.31	0.02	8.3%	33.2%
Conn. B.	0.25	0.01	75.8%	74.0%	0.19	0.02	52.4%	53.1%
Def. CC	0.23	0.03	87.6%	84.6%	0.21	-0.01	57.6%	73.0%
Mal. Ex.	0.05	-0.03	82.2%	87.9%	0.02	0.00	56.7%	64.9%
Iono	0.14	0.02	81.4%	78.5%	0.11	-0.03	50.2%	50.0%
LSVT	0.25	0.02	69.8%	68.5%	0.21	0.02	34.5%	32.0%
MAGIC	0.18	0.03	79.4%	89.4%	0.16	0.01	44.8%	79.5%
Ozone	0.22	0.03	73.7%	69.8%	0.18	0.04	44.8%	39.9%
Pima	0.13	0.02	52.2%	72.4%	0.13	0.00	23.9%	45.1%
Pl. Rel.	0.28	0.04	48.0%	49.2%	0.29	-0.01	17.6%	19.9%
QSAR	0.18	0.03	81.6%	80.7%	0.15	0.01	56.6%	61.0%
StatLog	0.15	0.02	87.5%	86.8%	0.13	0.00	58.9%	65.4%

than the proposed HE. Indeed, Table 2.8 does not include the configuration $N = 500$, as the amount of useless neurons comprehensibly grows asymptotically as N grows.

Numerical results show that -in terms of classification error- the gap between E-S and HE keeps small also when l_1 regularization is involved. Indeed, it is worth to note that the generalization performance of E-S did not change significantly with respect to the experiments involving the conventional l_2 regularization term (as per Table 2.7). Moreover, Table 2.8 proves that -given a dataset- the amount of pruned neurons was often larger in E-S than in HE. In a few cases, the gap was actually substantial. This in turn means that E-S and HE in general included a different amount of neurons with unfruitful configurations. Nonetheless, the rate of useless neurons was usually smaller in HE then in E-S. As a major result, this experimental session confirmed that HE can effectively balance generalization performance and size of the hidden layer. Such ability allows HE to make the most of a coarse activation function such as the hard-limit function.

2.5.3 Strong Similarity Function Heuristic Algorithm

Up to now, the proposed Algorithms 3, 4 targeted an hypothesis space where a different parameter χ (r in the scalar product's specific case) characterizes each neuron/remapping unit, coherently with ELM paradigm. This configuration is the most profitable one in term of computational cost, fixing N a priori, because the definition of a unique parameter requires a model selection step. On the other hand, subsection 2.3.1 showed

that the selection of a unique hyper-parameter can be advantageous in the case in which a good mapping can be defined all over the input domain.

With this goal, the definition of strongly (ϵ, γ)-good similarity function can be a good starting point. In fact, it can support a straightforward learning algorithm [27]. First, one draws a sufficiently large set of landmarks, $\mathcal{L}^{(+)}$, from samples belonging to class '+1', and a corresponding (sufficiently large) set, $\mathcal{L}^{(-)}$, of landmarks for class '-1'. An unseen input sample, then, can be ascribed to either category according to the following rule: the predicted class is '+1' if the test sample is, on average, closer to the elements of $\mathcal{L}^{(+)}$ than to the elements of $\mathcal{L}^{(-)}$, and vice versa.

Given a strongly (ϵ, γ)-good similarity function, it can be proved [27] that the above algorithm produces a predictor having bound $(\epsilon + \delta)$ on the classification error. Here, δ is a function of both γ and the number of landmarks, L:

$$\delta = 2 \cdot e^{-\frac{L\gamma^2}{16}} \tag{2.25}$$

The assumptions imposed by the definition of strong similarity function are clearly too stringent to fit with real world problems. However, it can set the basis to drive an efficient model selection process, under the assumption that a sufficiently large dataset is available.

The proposed algorithm provides an effective heuristic to set the quantity $\chi = \chi_1 = \chi_2 = ... = \chi_L$ in (2.11). As per equation (2.25), given a set of landmarks one aims to choose the value of χ yielding the largest γ; here, ϵ depends on the specific selection of the landmarks. Thus, given a training set $\mathcal{T}$ and the eventual set of landmarks, one can proceed as follows:

1. compute the average similarity between the samples of class '+1' and all the landmarks;
2. compute the average similarity between the samples of class '-1' and all the landmarks;
3. estimate γ by assessing the gap between the two quantities computed above.

By repeating such procedure for a set of candidate values of the hyper-parameter, one identifies the setting that gives the largest (estimated) value of γ. Remarkably, this procedure relies on large datasets to provide a statistically consistent estimate of γ.

Algorithm 5 outlines the overall algorithm, which can support any policy of landmarks selection (i.e., $\boldsymbol{r}_j \in \mathcal{T}$ or $\boldsymbol{r}_j \in \mathcal{X}$). Actually, the underlined hypothesis is that the

Algorithm 5 Fast model selection algorithm that exploits the definition of (ϵ, γ) strong similarity function

Input

- a labeled training set $\mathcal{T}^+ = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z^+\}$ $\mathcal{T}^- = \{(\boldsymbol{x}, y)_i; i = 1, ..., Z^-\}$
- number of neurons N
- a set of candidate values $\{\chi_j, j = 1, ..., J\}$
- a similarity function ϕ

0. **Initialize**
generate the set of random landmarks $\boldsymbol{r}_j, j = 1, ..., N$

1. **Similarities barycenters**
for $j = 1$ to J **do**
average similarity between $\mathcal{T}^+$ and $\mathcal{L}$

$$\boldsymbol{S}_j^+ = \frac{1}{Z^+} \sum_i \sum_n \phi(\boldsymbol{x}_i, \boldsymbol{r}_n, \chi_j), \boldsymbol{x} \in \mathcal{T}^+, \boldsymbol{r} \in \mathcal{L}$$

average similarity between $\mathcal{T}^-$ and $\mathcal{L}$

$$\boldsymbol{S}_j^- = \frac{1}{Z^-} \sum_i \sum_n \phi(\boldsymbol{x}_i, \boldsymbol{r}_n, \chi_j), \boldsymbol{x} \in \mathcal{T}^-, \boldsymbol{r} \in \mathcal{L}$$

end for
2. **Margin**
estimate γ for each χ
$\hat{\gamma}_j = |S_j^+ - S_j^-|, j = 1, ..., J$
3. **Selection**
$j^* = \max_j \hat{\gamma}_j$
remap all the patterns $\boldsymbol{x} \in \mathcal{T}$ by using the following mapping function

$$\boldsymbol{\psi}(\boldsymbol{x}) = \{\phi(\boldsymbol{x}, \boldsymbol{r}_1, \chi_{j^*}), ..., \phi(\boldsymbol{x}, \boldsymbol{r}_N, \chi_{j^*})\}$$

2. **Learning**
train a linear predictor in the space $\boldsymbol{\psi} : \mathcal{X} \to \mathbb{R}^N$

heuristic performs better as long as the similarity function is as *strong* as possible (as per definition Definition 2.[27]). This assumption cannot hold in general; nonetheless, the proposed algorithm is expected to yield outcomes that are reasonably close to that obtained with a standard model selection.

The proposed heuristic strategy, involves only one complete training iteration. Thus, its computational cost combines $\boldsymbol{O}_{TR}$ and the computational cost of Algorithm 5, which only depends on the assessment of the terms S^+ and S^-. Formally the overall computational cost of the heuristic procedure (HR) can be expressed as:

$$\boldsymbol{O}_{HR} \cong \alpha Z^2 \cdot N + \beta N^3 + J \cdot Z \cdot N \tag{2.26}$$

Under the reasonable assumption $Z >> N$, $\boldsymbol{O}_{MS}$ scales actually as $J \cdot Z^2$. In case of the proposed heuristic, though, the term J is a multiplier for a linear function of the term Z. This in turn means that if Z big enough the overload imposed by the proposed algorithm becomes negligible with respect to $\boldsymbol{O}_{TR}$, i.e., with respect to the computational cost of a standard training in ELM.

2.5.3.1 Experimental Results

The experimental session aimed at comparing the performance of the two available options for the setup of the mapping layer in the hypothesis space (2.11), namely, the speculative approach applied by basic ELM, and the selection of a single similarity notion supported by the novel algorithm. The assessment assumed a critical scenario characterized by limited computational resources; thus, one would have to limit the size of the mapping layer. Six benchmarks characterized by the presence of consistent quantity of data were used: Checkerboard [52], USPS [53], Forest Cover Type [54], SUSY [55], HIGGS [56] and Intrusion Detection [57]. The selected benchmarks were chosen from a different set with respect to the ones considered in the previous experiment campaigns. In this case, in accordance with the assumptions introduced above, the experimental campaign involved medium-big size dataset.

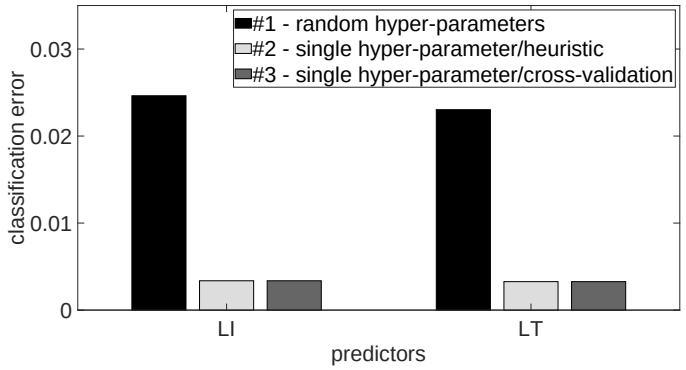
For each benchmark, four alternative configurations of the remapping layer were compared:

1. landmarks randomly drawn from $\mathbb{R}^D$ and random hyper-parameters (standard ELM).
2. landmarks randomly drawn from $\mathbb{R}^D$ and one hyper-parameter.
3. landmarks randomly drawn from $\mathcal{T}$ and random hyper-parameters.
4. landmarks randomly drawn from $\mathcal{T}$ and one hyper-parameter.

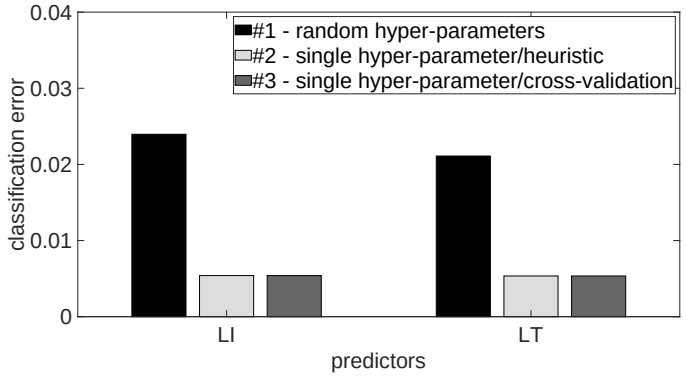
The configurations 3 and 4 basically modify the configurations 1 and 2, respectively, by changing the sampling domain to draw landmarks. In the case of configuration 2 and 4, two alternative methods were tested to set up the hyper-parameter: model-selection via cross-validation, and the proposed heuristic. As a result, the experimental session allowed to compare the outcomes of the heuristic with those of a standard (computationally demanding) approach to the hyper-parameter setting problem.

An RBF similarity function was adopted in all tests, and the linear separators in the remapped space were trained by solving the minimization problem (2.5). The allowed settings for the RBF functions were $\chi = \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$, whereas the regularization parameter varied in the set: $\lambda = \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$.

Checkerboard Dataset The Checkerboard dataset [52] is a synthetic, bi-class benchmark commonly used in large-scale learning problems. Samples lie in a 2-dimensional space. The first experiment involved a balanced training set with 200,000 patterns and a (balanced) test set with 20,000 patterns. The predictors were designed to include 2,000 neurons in the mapping layer; i.e., input data were remapped by using 2,000 landmarks (1% of the training set). The graph in Figure 2.8(a) reports on the results of this experiment: the y-axis gives the average classification error on the test set, expressed in the range [0,1] as the percentage of the overall test set; the average value is computed over 5 runs, i.e., 5 different randomizations of the mapping layer. On the x-axis, bars are grouped according to the source of landmarks: tag LI refers to predictors based on landmarks randomly drawn from the input domain $\mathbb{R}^D$; tag LT refers to predictors based on landmarks randomly drawn from the training set, $\mathcal{T}$. In each group, predictor 1 used random hyper-parameters; predictor 2 used a single hyper-parameter set according to the proposed heuristic; predictor 3 used a single hyper-parameter set by exploiting conventional cross-validation. The latter predictor completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Figure 2.8(a) provided some interesting outcomes. In each group, predictors 2 and 3 scored the same classification error. In turn, the proposed heuristic always competed with the conventional cross-validation. Moreover, in both groups LI and LT, the predictors based on the proposed heuristic improved over the classifiers using random hyper-parameter settings. Finally, the sampling domain of landmarks did not affect the performances of the predictors significantly. A second experiment involved a balanced training set with 20,000 patterns and a balanced test set holding 20,000 patterns. The predictors were again designed to include 2,000 neurons in the mapping layer; i.e., input data were remapped by using 2,000 landmarks (10% of the training set in this case). The graph in Figure 2.8(b) reports on the results of this experiment and follows the same conventions used in Figure 2.8(a). In Predictor 3, the cross-validation process used the training set and a balanced validation set including 20,000 patterns. Overall, this test confirmed the outcomes of the first one. Table 2.9 summarizes the results of the two experiments and gives the average classification errors (CEs) scored by each predictor, paired with the standard deviation between brackets. In addition, the Table provides, for predictors 2



(a)



(b)

FIGURE 2.8: Experimental session: Checkerboard dataset; (a) first experiment; (b) second experiment.

and 3, the values of χ selected for the proposed heuristic and the cross-validation procedure, respectively. For completeness, the bottom row of Table 2.9 gives the results scored by a standard ELM using the conventional sigmoid as the activation function. Empirical evidence confirmed that the predictors based on a single hyper-parameter according to the proposed heuristic outperformed the other options. It is worth noting that the gap between predictor 1 and predictors 2 and 3 is significantly bigger than one standard deviation. It is also worth noting that both the heuristic and the cross-validation method always converged to the same value ($\chi = 0.1$), independently of the experiment and of the sampling domain of landmarks.

Forest Cover Type Dataset The Forest Cover Type (CovType) dataset [54] holds patterns that describe the forest cover type described by cartographic variables. Samples lie in a 54-dimensional space. The dataset involves a multi-class problem, and this test addressed the bi-class problem "class 2 versus other classes". The first experiment involved a balanced training set with 250,000 patterns and a balanced test set with 20,000 patterns. The predictors were designed to include 2,500 neurons in the mapping

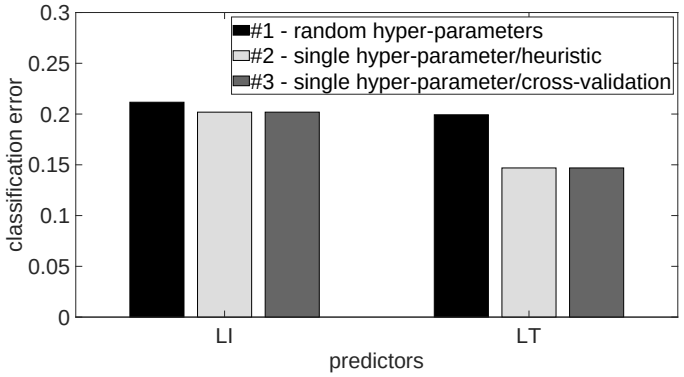
layer; i.e., input data were remapped by using 2,500 landmarks (1% of the training set). The graph in Figure 2.9(a) shows the results of this experiment by using the same format of Figure 2.8. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns.

Figure 2.9(a) again confirms that the proposed heuristic compared with the standard cross-validation. In addition, in both group LI and group LT the predictors supported by the proposed heuristic (predictor 2) improved over the classifiers using random hyper-parameters. In particular, predictor 2 obtained the best performance when exploiting landmarks drawn from the training set. Figure 2.9(b) reports on the result of the second experiment. In this case, a balanced training set with 25,000 patterns and a balanced test set with 20,000 patterns were involved. Predictors were designed to include 2,500 neurons in the mapping layer; i.e., input data were remapped by using 2,500 landmarks (10% of the training set). Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. This experiment validated the outcomes of experiment 1. Table 2.10 reviews the results of the two experiments. The predictor LT-2 (landmarks drawn from the training set, single hyper-parameter set according to the proposed heuristic) scored the best overall performance in both experiment 1 and experiment 2. Evidence witnessed a significant gap in performance as compared with predictor LT-1, predictor LI-1, and standard ELM with sigmoid activation function. Again, the gap between predictor 1 and predictors 2 and 3 is significantly bigger than a standard deviation. The proposed heuristic always agreed with full cross-validation on the values of χ. The predictors using landmarks from the input space (LI) converged to $\chi = 10$, while the predictors using landmarks from the training set (LT) converged to $\chi = 1$.

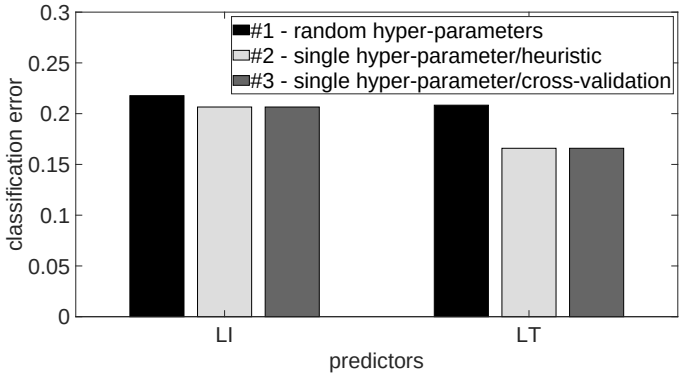
USPS Dataset The USPS dataset [53] is a popular benchmark that collects pattern obtained from the scanning of handwritten digits from envelopes by the U.S. Postal Service. The original size of the images is 16×16 pixels. This session inherited the setup

Table 2.9: Experimental session: Checkerboard dataset

	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.025 (0.002)	-	0.024 (0.002)	-
	#2	0.003 (0.000)	$1 \cdot 10^{-1}$	0.005 (0.000)	$1 \cdot 10^{-1}$
	#3	0.003 (0.000)	$1 \cdot 10^{-1}$	0.005 (0.000)	$1 \cdot 10^{-1}$
LT	#1	0.023 (0.001)	-	0.021 (0.003)	-
	#2	0.003 (0.000)	$1 \cdot 10^{-1}$	0.005 (0.000)	$1 \cdot 10^{-1}$
	#3	0.003 (0.000)	$1 \cdot 10^{-1}$	0.005 (0.000)	$1 \cdot 10^{-1}$
ELM sigmoid		0.118	-	0.121	-



(a)



(b)

FIGURE 2.9: Experimental session: CovType dataset; (a) first experiment; (b) second experiment

proposed in [52], which focused on the problem '0' versus '1'. To enlarge the original dataset, the images were first rescaled from 16×16 pixels to 26×26 pixels. Then, new patterns were generated by translating the original images in all directions, shifting ranged from two pixels to five pixels. Eventually, samples lay in a 676-dimensional space. The first experiment involved a balanced training set with 200,000 patterns and a balanced test set with 20,000 patterns. The predictors were designed to include 2,000 neurons in the mapping layer; i.e., input data were remapped by using 2,000 landmarks

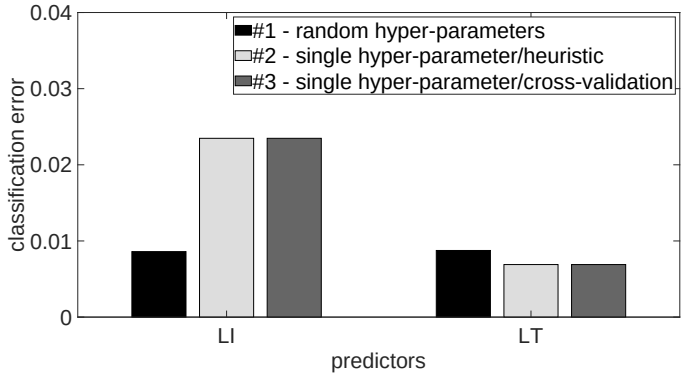
TABLE 2.10: Experimental session: CovType dataset

	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.212 (0.003)	-	0.218 (0.003)	-
	#2	0.202 (0.003)	$1 \cdot 10^1$	0.207 (0.002)	$1 \cdot 10^1$
	#3	0.202 (0.003)	$1 \cdot 10^1$	0.207 (0.002)	$1 \cdot 10^1$
LT	#1	0.199 (0.003)	-	0.208 (0.003)	-
	#2	0.147 (0.002)	1	0.166 (0.001)	1
	#3	0.147 (0.002)	1	0.166 (0.001)	1
ELM sigmoid		0.167	-	0.183	-

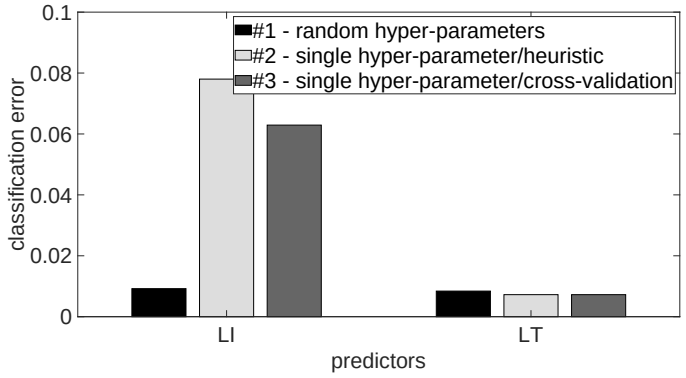
(1% of the training set). The graph in Figure 2.10(a) presents the results in the same format of Figure 2.8. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set holding 20,000 patterns. Figure 2.10(a) points out that the predictors based on landmarks drawn from the input space (LI) and those using landmarks from the training set (LT) performed differently. Again, the predictors supported by the proposed heuristic compared with the predictors supported by cross-validation. In the case of group LI, the best performance was scored by predictor 1, i.e., a basic ELM. Conversely, in group LT, predictor 2 (single hyper-parameter) slightly outperformed predictor 1 (random hyper-parameters). A similar trend characterized the second experiment. Here, a balanced training set with 20,000 patterns and a balanced test set with 20,000 patterns were involved. The predictors were designed to include 2,000 neurons in the mapping layer; i.e., input data were remapped by using 2,000 landmarks (10% of the training set in this case). The bar graph in Figure 2.10(b) reports on the results of this experiment. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Table 2.11 reviews the results of the two experiments. An interesting outcome is that predictor LT-2 (landmarks drawn from the training set, single hyper-parameter set by the proposed heuristic) scored the best overall performance in both experiments with a gap that is again bigger than a standard deviation. In addition, the proposed heuristic always identified the same value of that was prompted by full cross-validation. However, the predictors using landmarks drawn from the input space (LI) converged to $\chi = 100$, whereas the predictors using landmarks from the training set (LT) were set by using $\chi = 10$.

A non-marginal outcome of this experimental section regards the fact that both heuristic and model selection approaches perform poorly when the landmarks are set randomly. A possible explanation of this result is given by the fact that the probability of find a landmark belonging to the input domain eq. (2.13) is small. This in turns means that the training set is more concentrated in some part of the space. Both approaches 2 and 3 searches for a solution that is good in general, with poor results. Instead approach 1, randomly setting different notions of similarity for different portions of the space, even with some unfruitful configurations, find a set of effective combinations that enable a correct classification of the patterns.

SUSY Dataset The SUSY dataset [55] was set up to assess machine-learning approaches on problems that search for exotic particles in high-energy physics [58]. It involves a bi-class problem: processes where new supersymmetric particles are produced, versus a background process yielding the same detectable particles but fewer invisible particles and distinct kinematic features. Samples lie in an 18-dimensional space. The



(a)



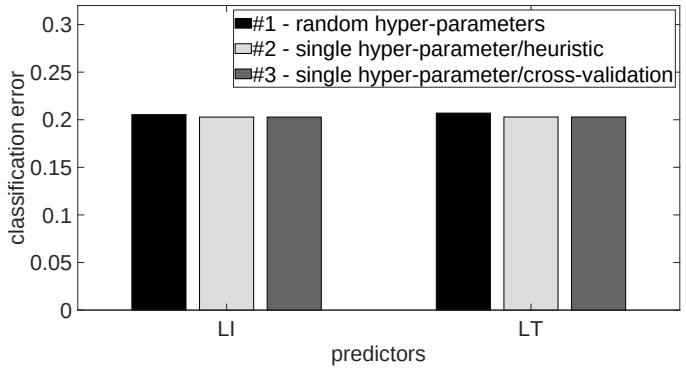
(b)

FIGURE 2.10: Experimental session: USPS dataset; (a) first experiment; (b) second experiment

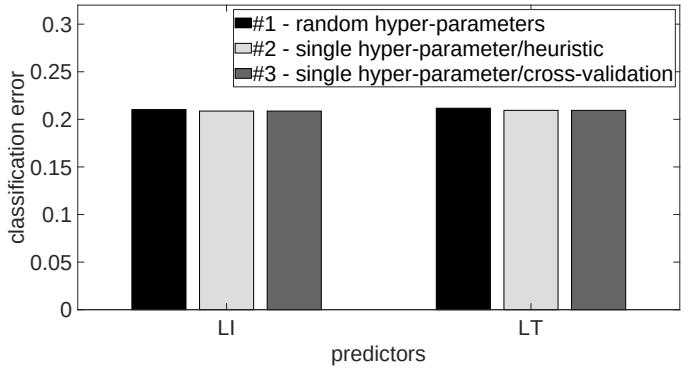
first experiment involved a balanced training set with 300,000 patterns and a balanced test set with 20,000 patterns. The predictors were designed to include 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (1% of the training set). The graph in Figure 2.11(a) gives the experimental results obtained. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Figure 2.11(a) shows that all predictors achieved the same performance. The same trend was found in the second experiment,

TABLE 2.11: Experimental session: usps dataset

	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.009 (0.000)	-	0.009 (0.000)	-
	#2	0.023 (0.001)	$1 \cdot 10^2$	0.078 (0.002)	$1 \cdot 10^2$
	#3	0.023 (0.001)	$1 \cdot 10^2$	0.063 (0.001)	$1 \cdot 10^1$
LT	#1	0.009 (0.001)	-	0.008 (0.001)	-
	#2	0.007 (0.000)	$1 \cdot 10^1$	0.007 (0.000)	$1 \cdot 10^1$
	#3	0.007 (0.000)	$1 \cdot 10^1$	0.007 (0.000)	$1 \cdot 10^1$
ELM sigmoid		0.009	-	0.009	-



(a)



(b)

FIGURE 2.11: Experimental session: SUSY dataset; (a) first experiment; (b) second experiment.

where a balanced training set with 30,000 patterns and a balanced test set with 20,000 patterns were involved. The predictors were designed to include 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (10% of the training set). Figure 2.11(b) reports on the result of the second experiment; Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Table 2.12 summarizes empirical evidence that all the predictors scored the same classification error. Moreover, both the heuristic and the cross-validation always agreed on the same value (χ= 10), independently of the experiment and of the sampling domain of landmarks.

HIGGS Dataset The HIGGS dataset [56], just like the SUSY dataset, is used to evaluate the performance of machine-learning approaches in high-energy physics [58]. It involves a bi-class problem: processes where new theoretical Higgs bosons are produced versus a background process featuring the same decay products but distinct kinematic features. Samples lie in a 28-dimensional space. The first experiment again involved a balanced training set with 300,000 patterns and a balanced test set with 20,000 patterns.

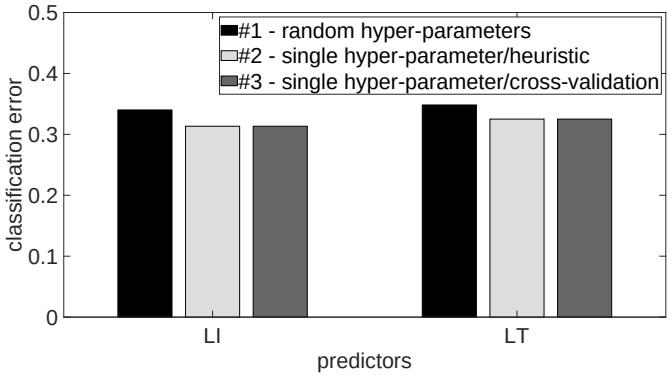
The predictors were designed to include 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (1% of the training set). The graph in Figure 2.12(a) presents the results by using the same format of Figure 2.8. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Figure 2.12(a) confirmed that, for this dataset too, the proposed heuristic compared with the basic cross-validation. In both group LI and group LT the predictors based on the proposed heuristic (predictor 2) improved over the classifiers using random hyper-parameters. Similar results were achieved in the second experiment, involving a balanced training set holding 30,000 patterns and a balanced test set with 20,000 patterns. The predictors included 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (10% of the training set). Figure 2.12(b) reports on the result of this second experiment; Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Table 2.13 reviews the results of both experiments, and confirms that predictor 2 always outperformed predictor 1 and independently of the sampling strategy of landmarks. A remarkable outcome was that the best overall performance was always attained by predictor LI-2, i.e., the classifier based on landmarks drawn from the input space. The Table also shows that both the heuristic and the cross-validation always converged to the same setting of the hyper-parameter, $\chi = 10$, independently of the experiment and of the sampling strategy of landmarks.

Table 2.12: Experimental session: SUSY dataset

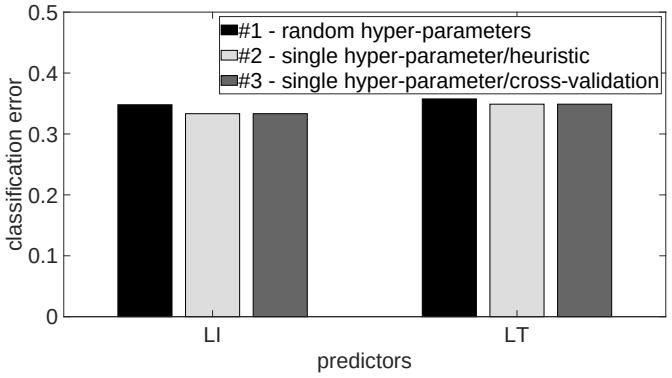
	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.205 (0.002)	-	0.210 (0.004)	-
	#2	0.203 (0.002)	$1 \cdot 10^1$	0.209 (0.005)	$1 \cdot 10^1$
	#3	0.203 (0.002)	$1 \cdot 10^1$	0.209 (0.005)	$1 \cdot 10^1$
LT	#1	0.207 (0.003)	-	0.212 (0.004)	-
	#2	0.203 (0.003)	$1 \cdot 10^1$	0.209 (0.005)	$1 \cdot 10^1$
	#3	0.203 (0.003)	$1 \cdot 10^1$	0.209 (0.005)	$1 \cdot 10^1$
ELM sigmoid		0.201	-	0.213	-

Table 2.13: Experimental session: HIGGS dataset

	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.340 (0.002)	-	0.348 (0.002)	-
	#2	0.313 (0.001)	$1 \cdot 10^1$	0.333 (0.002)	$1 \cdot 10^1$
	#3	0.313 (0.001)	$1 \cdot 10^1$	0.333 (0.002)	$1 \cdot 10^1$
LT	#1	0.348 (0.002)	-	0.358 (0.001)	-
	#2	0.325 (0.002)	$1 \cdot 10^1$	0.349 (0.001)	$1 \cdot 10^1$
	#3	0.325 (0.002)	$1 \cdot 10^1$	0.349 (0.002)	$1 \cdot 10^1$
ELM sigmoid		0.351	-	0.364	-



(a)



(b)

FIGURE 2.12: Experimental session: HIGGS dataset; (a) first experiment; (b) second experiment

Intrusion Detection The intrusion detection dataset introduced in [57] consists of a collection of 5 days of a network traffic where seven common type of attack are performed; each day a different set of attacks is performed. In the experimental section the traffic generated on Wednesday is analyzed, because it presents the biggest number of attacks; The problem is approached as a binary classification, where the target is the distinction between "benign" and "attacks". Following feature extraction process described by authors of the original paper [57] a total of 691416 patterns with 80 dimension each have been extracted.

The first experiment again involved a balanced training set with 200,000 patterns and a balanced test set with 20,000 patterns. The predictors were designed to include 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (1% of the training set). The graph in Figure 2.13(a) presents the results by using the same format of Figure 2.8. Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns.

Figure 2.13(a) confirmed the effectiveness of the proposed heuristic compared with the

basic cross-validation. In group LI the predictors based on the proposed heuristic (predictor 2) improved over the classifiers using random hyper-parameters, while the best score is achieved by classifier 1 with landmark sampled from the training set.

Similar results were achieved in the second experiment, involving a balanced training set holding 20,000 patterns and a balanced test set with 20,000 patterns. The predictors included 3,000 neurons in the mapping layer; i.e., input data were remapped by using 3,000 landmarks (10% of the training set). Figure 2.13(b) reports on the result of this second experiment; Predictor 3 completed the cross-validation process by using the training set and a balanced validation set including 20,000 patterns. Table 2.14 reviews the results of both experiments, and confirms that predictor of group LT outperform the ones of group LI, independently by hyper parameter selection. The Table also shows that both the heuristic and the cross-validation always converged to the same setting of the hyper-parameter, $\chi = 10$, for group LI and $\chi = 1$ for group LT.

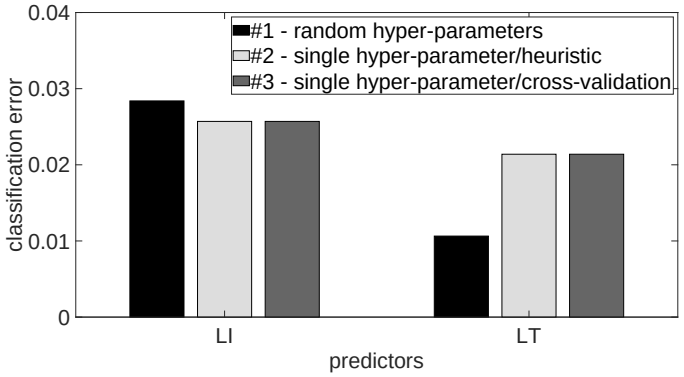
2.5.3.2 Concluding Remarks

In general, experimental results proved that the proposed approach to the setup of the mapping layer could attain satisfactory outcomes when dealing with medium-large datasets. The assessment focused on a challenging scenario, in which limited computational resources are available. Accordingly, the experiments compared predictors that should satisfy a tight constraint in the size of the mapping layer. Overall, the predictors based on the proposed heuristic always proved able to improve over the standard ELM model. In fact, the SUSY dataset provided the only case in which the improvement was not substantial. Conversely, the Intrusion dataset provided the only case in which the adoption of a single notion of similarity did not lead to the best predictor.

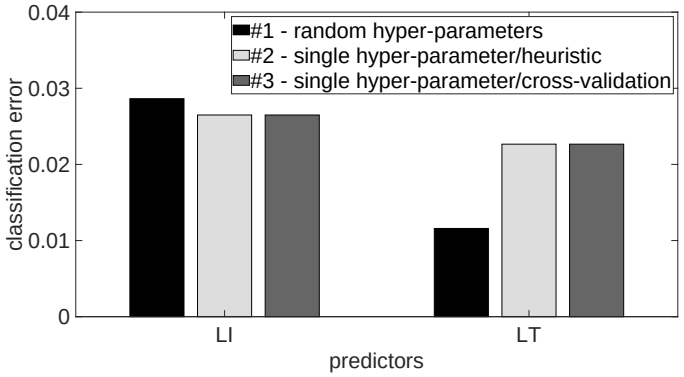
Thus, the proposed reinterpretation of the standard ELM model can lead to paradigm that better balances generalization ability and size of the mapping layer. Such aspect is indeed crucial when dealing with large datasets, as the latter parameter has an impact

Table 2.14: Experimental session: Intrusion detection dataset

	Predictor	Experiment 1		Experiment 2	
		CE	χ	CE	χ
LI	#1	0.028 (0.000)	-	0.029 (0.000)	-
	#2	0.026 (0.000)	$1 \cdot 10^1$	0.026 (0.000)	$1 \cdot 10^1$
	#3	0.026 (0.000)	$1 \cdot 10^1$	0.026 (0.000)	$1 \cdot 10^1$
LT	#1	0.011 (0.001)	-	0.012 (0.001)	-
	#2	0.021 (0.002)	1	0.023 (0.000)	1
	#3	0.021 (0.002)	1	0.023 (0.000)	1
ELM sigmoid		0.059	-	0.062	-



(a)



(b)

FIGURE 2.13: Experimental session: Intrusion dataset; (a) first experiment; (b) second experiment

on the computational resources eventually involved both in the training phase and in the implementation of the predictor.

2.5.4 Comparative Analysis

As said in the introduction, the proposed algorithms (3, 4, 5) shares some common points. In fact, all these algorithms involve a supervised selection strategy to the setup of the hidden parameter χ. This strategy has been selected because it does not require to recompute the metric $M(\boldsymbol{x}, \boldsymbol{r})$ that involves computationally demanding vector to vector operations (see eq. (2.15)). On the other hand, each of those approaches presents some peculiarities that makes it more suitable for an application scenario respect to another.

Algorithm 4 has been designed to handle the peculiarities of the threshold function. This activation function in fact admits only two values and for this reason, it is more prone to pathological configurations respect to standard analogical activations such as SGM or RBF. At the same time, it leads to extremely efficient digital implementations.

For this reason, even if, in principle, it is possible to apply algorithms 3 and 5 for free parameterss selection, they result less effective, given the binary nature of the mapping. As a consequence, a specific selection criterion has been proposed.

Algorithms 3, 5 refer to the same merit function. However, few major differences characterize the proposed methodologies. Algorithm 5 targets the selection of a single value of the shape parameter for all the mapping units. This value is selected among a pool of candidate. Instead, Algorithm 3 limits its action to the reject of pathological configurations. This difference underlines different hypothesis: in case of algorithm 2.18, the assumption is that exists a simple similarity notion ϕ that is good all over the input domain, while in the case of algorithm 3 the requirement is that exists a set of functions that are good for some portions of the input domain. A second major difference between the two proposed approaches is that the computational load of procedure 5 is fixed a priori, while algorithm 3 applies a random selection criterion that involves an unknown number of steps for the selection of the random parameter of each neuron. Finally, Algorithm 5 requires a relatively big dataset to estimate the consistency of strong similarity function hypothesis, while, in principle, algorithm 3 does not set explicitly any constraint to the size of the training set.

The literature indeed provides several works that dealt with the construction of the mapping layer in ELM. In general, the goal is to define a strategy that can balance generalization ability and number of neurons in the eventual predictor. In [40, 41] the optimization problem has been slightly modified to the purpose of favoring sparse solution, thus removing ineffective neurons. In [41] the authors exploited a l_1 regularization term to remove ineffective neurons; then, the selected neurons were involved in the training of a standard ELM. Optimally pruned ELM [40] relies on a combination of multiple sparse regressions and leave-one-out mechanism to prune the less informative neurons. In both the papers the pruning process is computationally demanding and adds a few hyper-parameters to the optimization problem.

Biologically-inspired optimization has indeed stimulated various works: self-adaptive evolutionary ELM [42], dolphin swarm ELM [59], genetic ensemble of ELM [60], particle swarm optimization based ELM [43] and Artificial Immune System based ELM [61]. All these approaches share the common idea of exploiting strategies derived from the observation of natural phenomena to enhance the performance of the machines in term of generalization ability. In general, such models are computationally demanding. In fact, most of the them involve non-convex optimization problems.

In [62, 63] the target was the implementation of the predictor on resource-constrained device. In practice, only these two works are comparable in term of additive computational cost respect to the ones introduced in this Thesis. Hence, the underlying mapping

strategy was designed to fulfill specific constraints on the admissible activation function, making these works less general than the proposed one.

Chapter 3

Tensor Learning

Tensors provide a more convenient way to describe multi-way data and to suitably capture their multi-linear structure. This basic aspect becomes relevant when dealing with machine learning (ML): multi-spectral images [17], computer vision [38], authorship identification [64] and multi-sensory arrays [65, 66] provide relevant examples of domains in which ML-based predictors are required to deal with multi-way data. Accordingly, one needs specific learning methods that can explicitly exploit the multi-mode relations that connect the various data associated with one pattern. These specialized methods [67–70] can outperform conventional approaches, which remap multi-way data into a classical vector representation for the subsequent application of ML models. In this regard, the crucial point is that the geometrical information inherently embedded in the tensor structure can be exploited to boost the learning process, i.e., to improve convergence speed. Conversely, traditional models based on vector space -which actually ensures universal approximation capabilities- face the risk of not capturing such geometrical information. As a major consequence, the convergence speed can be substantially different.

This chapter addresses the use of the similarity function framework and shows that this theory [27] can stimulate a novel design strategy, based on algorithm 2, which may represent an alternative to standard kernel-based methods or SLFN. The same two main reasons of chapter 2 justify the focus on tensor-similarity-based models: first, these frameworks usually better succeed in balancing the generalization performance and the computational complexity of the trained predictors. This is a crucial aspect when implementing the classifiers on embedded systems. Secondly, similarity-based models provide a viable option for avoiding the computationally-intensive learning procedures that affect iterative training.

The literature provides several tensor-oriented ML frameworks, which might be categorized into two groups.

"Projections learning" approaches focus on multilinear projections, to remap tensor data down to a lower-dimensional space. The underlying goal is to find a projection schema that can capture the (unknown) structure of the problem at-hand, so that one may eventually apply ML methods to learn a decision function in the lower-dimensional space. Those methods mostly address the problem of finding the projection that satisfies an optimality criterion [71–73], with the partial exception of [74], which relies on higher-order random projections.

Conversely, "function learning" frameworks aim to learn a decision function that is designed to accept tensors as inputs. The related training algorithms often involve an iterative process [75–78] to solve convex optimization problems; training stops when some pre-set condition is fulfilled. Otherwise, training procedures rely on standard convex optimization; this case includes support higher-order tensor machines (SHTMs) [79, 80] and the kernel-based framework proposed by Signoretto et al. [81], based on a tensor-oriented kernel.

To achieve this goal, the present thesis introduces a similarity function that can suitably process multi-way data; thus, it computes the similarity between a given sample and a given landmark that lie in a tensor space. The function embeds a similarity notion that involves a three-step processing. First, multi-linear singular value decomposition (MLSVD) [82] is used to represent the landmark tensor as a set of orthonormal bases with their associate eigenvalues. Then the sample tensor is projected according to the landmark bases, to emphasize the intrinsic structural differences between the sample and the landmark. Finally, similarity is assessed by a metric function, which processes the eigenvalues of both the landmark and the (projected) sample.

The design of a suitable similarity function that can process patterns represented as tensors is the major challenge in tensor oriented approaches [80]. Present work characterizes both the geometrical and the cognitive properties of the proposed similarity notion, which indeed inherently embeds noise filtering capabilities. Such attribute makes the similarity function particularly suitable for applications that may suffer from noisy acquisition processes. Besides, this work shows that the availability of such similarity function leads to a tensor-oriented ML framework that features the following advantages:

- a training algorithm that does not require iterative processes, unlike most of the state-of-the-art function learning frameworks.
- a prediction function that -in terms of computational cost- bests the prediction function supported by a state-of-the-art kernel-based framework for tensorial data

[81]; such aspect becomes relevant when considering that a major goal of the thesis consist in the implementations of machine learning algorithms on resource-constrained devices [9].

- a predictor that compares positively with state-of-the-art tensor-oriented ML frameworks in terms of classification accuracy.

The chapter is structured as follows: Section 3.1 briefly recalls the basics about tensors; following Sections 3.2 and 3.3 respectively presents the proposed notion of similarity and the complete framework base on theory of learning with similarity functions. The comparison with state-of-the-art is presented in section 3.4. Finally, section 3.5 depict the outcomes of the experimental campaign.

3.1 Basic Operations

This section briefly review the necessary background for the comprehension of the proposed work.

Tensors represents an extension of vectors and matrices that can store information more conveniently. Formally, a tensor can be formalized as a multi-way array of numbers. In the following, $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_R}$ will refer to a real-valued tensor of order R. Accordingly, a vector is a tensor of order 1 and a matrix can be regarded as tensors of 2.

3.1.1 Tensor Unfolding

Unfolding is the process of reordering the elements of a tensor $\boldsymbol{\mathcal{T}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_R}$ into a matrix $\boldsymbol{T} \in \mathbb{R}^{P \times Q}$, where $P \times Q = I_1 \times I_2 \times \ldots \times I_R$. The notation $\boldsymbol{\mathcal{T}}_{(r)}$ will denote the mode-r unfolding of $\boldsymbol{\mathcal{T}}$. Accordingly, $\boldsymbol{\mathcal{T}}_{(r)} \in \mathbb{R}^{I_r \times I_1 I_2 I_{r-1} I_{r+1} \ldots I_R}$ and the reordering process follows the scheme exemplified in Fig. 3.1. In the example, a third-order tensor $\boldsymbol{\mathcal{T}}$ is unfolded in the matrices $\boldsymbol{\mathcal{T}}_{(1)}$, $\boldsymbol{\mathcal{T}}_{(2)}$, and $\boldsymbol{\mathcal{T}}_{(3)}$. The mode-1 unfolding $\boldsymbol{\mathcal{T}}_{(1)}$ is obtained by first partitioning the original tensor according to the slices $\boldsymbol{\mathcal{T}}(i_1, :, :)$. Then, the eventual matrix is created by placing side by side the slices in the original order. Likewise, $\boldsymbol{\mathcal{T}}_{(2)}$ and $\boldsymbol{\mathcal{T}}_{(3)}$ are obtained by partitioning the original tensor according to the slices $\boldsymbol{\mathcal{T}}(:, i_2, :)$ and $\boldsymbol{\mathcal{T}}(:, :, i_3)$, respectively. Such scheme can be easily extended to a generic tensor of order R.

Finally, $\boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{T}} \times_r \boldsymbol{B}$ will denote the mode-r product of $\boldsymbol{\mathcal{T}}$ and $\boldsymbol{B} \in \mathbb{R}^{J_r \times I_r}$; this product yields a tensor $\boldsymbol{\mathcal{Q}} \in \mathbb{R}^{I_1 \times I_2 \times I_{r-1} \times J_r \times I_{r+1} \times I_R}$, whose entries can be expressed as

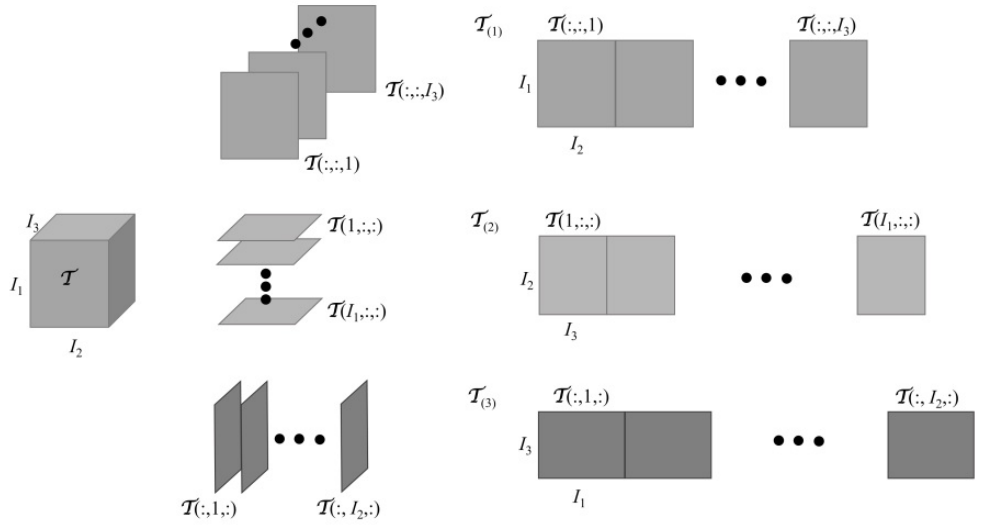


FIGURE 3.1: Unfolding of a third-order tensor.

$$q_{i_1 \ldots i_{r-1} j_r i_{r+1} \ldots i_R} = \sum_{i_r=1}^{I_r} t_{i_1 \ldots i_{r-1} i_r i_{r+1} \ldots i_R} b_{j_r i_r} \tag{3.1}$$

As a consequence, $\boldsymbol{\mathcal{Q}} = \boldsymbol{\mathcal{T}} \times_r \boldsymbol{B}$ can also be rewritten as $\boldsymbol{\mathcal{Q}}_{(r)} = \boldsymbol{B}\boldsymbol{\mathcal{T}}_{(r)}$.

3.1.2 Multi-Linear Singular Value Decomposition

MLSVD is a multi-linear generalization of the well-known singular value decomposition (SVD), supported by the model originally introduced in the Tucker decomposition [82, 83].

MLSVD represents a tensor $\boldsymbol{\mathcal{T}}$ as a multi-linear transformation of a core tensor, $\bar{\boldsymbol{\mathcal{T}}} \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_R}$, by means of the matrices $\boldsymbol{T}_r = [\boldsymbol{t}_1^{(r)} \boldsymbol{t}_2^{(r)} \ldots \boldsymbol{t}_{I_r}^{(r)}] \in \mathbb{R}^{I_r \times I_r} (r = 1, \ldots, R)$; here, the column vectors $\boldsymbol{t}_{I_r}^{(r)}$ define an orthonormal basis in $\mathbb{R}^{I_r}$. As a result,

$$\boldsymbol{\mathcal{T}} = \bar{\boldsymbol{\mathcal{T}}} \times_1 \boldsymbol{T}_1 \times_2 \boldsymbol{T}_2 \ldots \times_R \boldsymbol{T}_R \tag{3.2}$$

Each orthonormal basis, $\boldsymbol{T}_r$, can be obtained by applying conventional SVD to $\boldsymbol{\mathcal{T}}_{(r)}$. Hence, one has

$$\boldsymbol{\mathcal{T}}_{(r)} = \boldsymbol{U}_r \boldsymbol{\Sigma}_r \boldsymbol{V}_r^t \tag{3.3}$$

with $\boldsymbol{T}_r = \boldsymbol{U}_r$. Indeed, the core tensor $\bar{\boldsymbol{\mathcal{T}}}$ can be computed as follows:

$$\bar{\boldsymbol{\mathcal{T}}} = \boldsymbol{\mathcal{T}} \times_1 \boldsymbol{T}_1^t \times_2 \boldsymbol{T}_2^t ... \times_R \boldsymbol{T}_R^t \tag{3.4}$$

where $\bar{\boldsymbol{\mathcal{T}}}$ is an all-orthogonal and ordered tensor [82]. "All-orthogonal" means that all the rows of any mode-n unfolding of $\bar{\boldsymbol{\mathcal{T}}}$, $\bar{\boldsymbol{\mathcal{T}}}_{(r)}$, are mutually orthogonal. "Ordered" means that the following condition holds for such rows:

$$\|\bar{\boldsymbol{\mathcal{T}}}_{(r)}(1,:)\| \leq \|\bar{\boldsymbol{\mathcal{T}}}_{(r)}(2,:)\| \leq ... \leq \|\bar{\boldsymbol{\mathcal{T}}}_{(r)}(I_r,:)\| \tag{3.5}$$

It is worth noting that the Frobenius norm $\|\bar{\boldsymbol{\mathcal{T}}}_{(r)}(i_r,:)\|$ corresponds to the i_r-th r-mode eigenvalue of $\boldsymbol{\mathcal{T}}$, i.e., the i_r-th eigenvalue of $\boldsymbol{\mathcal{T}}_{(r)}$ [82]. Thus,

$$(\sigma_{i_r})_{(r)}^{\boldsymbol{\mathcal{T}}} = \boldsymbol{\Sigma}_r(i_r, i_r) = \|\bar{\boldsymbol{\mathcal{T}}}_{(r)}(i_r,:)\| \tag{3.6}$$

In addition, the columns of $\boldsymbol{T}_r$ are the r-mode eigenvectors of $\boldsymbol{\mathcal{T}}$. Furthermore, it holds:

$$\|\boldsymbol{\mathcal{T}}_{(r)}\|^2 = \|\bar{\boldsymbol{\mathcal{T}}}_{(r)}\|^2 = \sum_{i}^{I_r} \left((\sigma_{i_r})_{(r)}^{\boldsymbol{\mathcal{T}}}\right)^2 \tag{3.7}$$

3.1.3 Multi-Linear Extremal Energy

The concept of extremal energy plays a crucial role in SVD-based signal separation algorithms.

Definition 4. [34](Extremal directions of oriented energy): Let $\boldsymbol{A}$ be a $m \times r$ matrix with SVD $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^t$ where $\boldsymbol{\Sigma} = \texttt{diag}\{\sigma_i\}$. Then each direction of extremal oriented energy is generated by a left singular vector u_i with extremal energy equal to the corresponding singular value σ_i^2.

Definition 4 states that all the energy contained in a matrix can be described by means of the pairs (eigenvector, eigenvalues). In practice, the extremal directions of oriented energy express the hidden geometrical properties of the matrix. These properties indeed lead to the principle of optimal reconstruction in the minimal square error sense [34].

The following definition paves the way to the extension of the concept of extremal directions of oriented energy to the multivariate case.

Definition 5. [82](n-mode oriented energy): the n-mode oriented energy of an Rth-order tensor $\boldsymbol{\mathcal{T}}$ in the direction of a unit-norm vector $\boldsymbol{x}$, denoted by $OE_r(\boldsymbol{x}, \boldsymbol{\mathcal{T}})$, is the oriented energy of the set of r-mode vectors, i.e.,

$$OE_r(\boldsymbol{x}, \boldsymbol{\mathcal{T}}) \doteq \|\boldsymbol{x}^t \boldsymbol{\mathcal{T}}_{(r)}\|^2 \tag{3.8}$$

Definition 5 actually allows one to reinterpret Equation (3.7) according to the following property:

Property 3. [82](oriented energy). The directions of extremal r-mode oriented energy correspond to the r-mode singular vectors, with extremal energy value equal to the corresponding squared r-mode singular value.

Thus,

$$\|\boldsymbol{\mathcal{T}}_{(r)}\|^2 = \|\bar{\boldsymbol{\mathcal{T}}}_{(r)}\|^2 = \sum_{i}^{I_r} \left((\sigma_{i_r})^{\mathcal{T}}_{(r)}\right)^2 = \sum_{i}^{I_r} OE_r(\boldsymbol{U}_{(r)}(i_r, :), \boldsymbol{\mathcal{T}}) \tag{3.9}$$

As a result, let $\hat{\boldsymbol{\mathcal{T}}}$ be a tensor obtained by removing from $\boldsymbol{\mathcal{T}}$ the components of the r-mode vectors in the direction of an r-mode singular vector $\boldsymbol{U}_{(r)}(I_r, :)$ (i.e., the singular vector that corresponds to the smallest r-mode singular value). One has that

$$\|\boldsymbol{\mathcal{T}} - \hat{\boldsymbol{\mathcal{T}}}\|^2 = (\sigma^{(r)}_{I_r})^2 = OE_r(\boldsymbol{U}_{(r)}(I_r, :), \boldsymbol{\mathcal{T}}) \tag{3.10}$$

In general, $\hat{\boldsymbol{\mathcal{T}}}$ is not the best approximation in a square error sense. However, this approximation involves an error that is known a-priori, i.e., $((\sigma_{I_r})^{\mathcal{T}}_{(r)})^2$. Equation (3.10) proves that the MLSVD actually provides a detailed geometrical description of the energy distribution over the r-mode vector space. In practice, for each mode, the associate matrix defines the orthogonal directions in which the energy is more propagated.

3.2 Tensor Input Similarity Function

As shown in the previous chapter, the geometrical properties of the input space play a crucial role in the development of effective remapping layers. In general, the curse of dimensionality problem becomes more prominent when multi-linear input domains are considered and an extensive sampling of the input space becomes almost impossible. The aim of the following section is the development of a metric that can efficiently extract local information of the data manifold from available patterns exploiting all the geometrical information embedded in the interactions between different modes.

3.2.1 Extremal Energy Based Similarity Notion

The definition of a notion of similarity between a generic multi-way pattern and a multi-way landmark is the first step in designing the tensor-based similarity function. For the sake of clarity, to exemplify the rationale behind the proposed similarity notion and without loss of generality, let a landmark, $\boldsymbol{\mathcal{L}}$, be a tensor of order 2: $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{I_1 \times I_2}$. As per equation (3.2), $\boldsymbol{\mathcal{L}}$ can be represented as:

$$\boldsymbol{\mathcal{L}} = \bar{\boldsymbol{\mathcal{L}}} \times_1 \boldsymbol{L}_1 \times_2 \boldsymbol{L}_2 \tag{3.11}$$

The properties of the SVD also allow to rewrite $\boldsymbol{\mathcal{L}}$ as

$$\boldsymbol{\mathcal{L}} = \sum_{i=1}^{p} \boldsymbol{\Sigma}(i,i) \cdot \boldsymbol{u}_i \otimes \boldsymbol{v}_i = \sum_{i=1}^{p} \sigma_i^{\boldsymbol{L}} \hat{\boldsymbol{L}}_i \tag{3.12}$$

where p is the rank of $\mathcal{L}$, the eigenvalues σ_i are ordered in decreasing level of magnitude, and the matrices $\hat{\boldsymbol{L}}_i$ define an orthonormal basis in $\mathbb{R}^{I_1 \times I_2}$ for $\mathcal{L}$. In general, a matrix is univocally identified by the combination of eigenvalues σ_i and the corresponding orthonormal basis. Moreover, the matrices $\hat{\boldsymbol{L}}_i$ identify the most informative basis for the original matrix in a square-error sense. As a result, the following provides an optimal approximation of $\boldsymbol{\mathcal{L}}$ (in a square-error sense).

$$\hat{\boldsymbol{\mathcal{L}}} = \sum_{i=1}^{q} \sigma_i^{\boldsymbol{L}} \hat{\boldsymbol{L}}_i, \;\; \textit{with } q < p, \tag{3.13}$$

Conventional approaches to the assessment of the similarity between a generic pattern $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2}$ and a landmark, $\boldsymbol{\mathcal{L}}$, usually rely on some metrics that compares the eigenvalues and the orthonormal basis of $\boldsymbol{\mathcal{X}}$, $\{\sigma_i^{\boldsymbol{X}}, \hat{\boldsymbol{X}}_i\}$, with the eigenvalues and the orthonormal basis of $\boldsymbol{\mathcal{L}}$, $\{\sigma_i^{\boldsymbol{L}}, \hat{\boldsymbol{L}}_i\}$. These approaches would require the computation of two SVDs; instead, the similarity notion that is proposed in this thesis exploits the basis of $\boldsymbol{\mathcal{L}}$ as a reference, thus projecting the pattern $\boldsymbol{\mathcal{X}}$ on the most informative set of basis for the landmark. Accordingly, similarity is assessed by comparing the eigenvalues $\sigma_i^{\boldsymbol{L}}$ of $\boldsymbol{\mathcal{L}}$ with the *pseudo*-eigenvalues of $\boldsymbol{\mathcal{X}}$. The latter quantities can be formalized as the coefficients $\tilde{\sigma}_i^{\boldsymbol{X}}$ that support the following reconstruction of $\boldsymbol{\mathcal{X}}$:

$$\boldsymbol{\mathcal{X}} = \sum_{i=1}^{p} \tilde{\sigma}_i^{\boldsymbol{X}} \hat{\boldsymbol{L}}_i, \tag{3.14}$$

Extending this approach to tensors of any order is feasible, under the hypothesis that the landmark $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_R}$ has a core tensor $\bar{\boldsymbol{\mathcal{L}}}$ with non-null elements only for $i_1 = i_2 = ...I_r$. In general, though, this condition does not hold. Then, one can represent $\boldsymbol{\mathcal{L}}$ either by relying on (3.2), or as

$$\boldsymbol{\mathcal{L}} = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} ... \sum_{i_R=1}^{I_R} \bar{\boldsymbol{\mathcal{L}}}(i_1, i_2, ..., i_R) \cdot \boldsymbol{l}_{i_1}^{(1)} \otimes \boldsymbol{l}_{i_2}^{(2)} \otimes ... \otimes \boldsymbol{l}_{i_R}^{(I_R)} = \sum_{i=1}^{I} \sigma_i^{\mathcal{L}} \hat{\mathcal{L}}_i \ \textit{with}\ I = I_1 I_2 ... I_R. \quad (3.15)$$

In eq. (3.15), $\boldsymbol{l}_{i_r}^{(r)}$ represents the i_r-th column vector of the orthonormal basis $\boldsymbol{L}_r$; the resulting tensors $\hat{\boldsymbol{\mathcal{L}}}_i$ define a basis in $\mathbb{R}^{I_1 \times I_2 \times ... \times I_R}$. Equation (3.15) shows that to project a generic pattern $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2 \times ... \times I_R}$ onto the bases of $\boldsymbol{\mathcal{L}}$ one needs to align the corresponding bases $\{\boldsymbol{X}_r; r = 1, ..., R\}$ with the bases $\{\boldsymbol{L}_r; r = 1, ..., R\}$. According, let $\tilde{\boldsymbol{\mathcal{X}}}$ be a *pseudo*-core tensor obtained by the multilinear transformation of $\boldsymbol{\mathcal{X}}$ by the orthonormal bases $\boldsymbol{L}_r$, i.e., the bases that characterizes $\boldsymbol{\mathcal{L}}$:

$$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{L}_1^t \times_2 \boldsymbol{L}_2^t ... \times_R \boldsymbol{L}_R^t \quad (3.16)$$

$\tilde{\boldsymbol{\mathcal{X}}}$ is a *pseudo*-core tensor since it cannot be considered the core tensor of $\boldsymbol{\mathcal{X}}$ in a rigorous sense. According to equation (3.4), the core tensor $\bar{\boldsymbol{\mathcal{X}}}$ is obtained from the associate orthonormal basis, $\boldsymbol{X}_r$:

$$\bar{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{X}_1^t \times_2 \boldsymbol{X}_2^t ... \times_R \boldsymbol{X}_R^t \quad (3.17)$$

Anyway, $\tilde{\boldsymbol{\mathcal{X}}}$ can be very useful if one addresses the goal of assessing the similarity between $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$. In this regard, it is convenient to rewrite (3.16) as

$$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{L}_1^t \times_2 \boldsymbol{L}_2^t ... \times_R \boldsymbol{L}_R^t = \bar{\boldsymbol{\mathcal{X}}} \times_1 \boldsymbol{X}_1 \times_2 \boldsymbol{X}_2 ... \times_R \boldsymbol{X}_R \times_1 \boldsymbol{L}_1^t \times_2 \boldsymbol{L}_2^t ... \times_R \boldsymbol{L}_R^t = \bar{\boldsymbol{\mathcal{X}}} \times_1 \boldsymbol{L}_1^t \boldsymbol{X}_1 \times_2 \boldsymbol{L}_2^t \boldsymbol{X}_2 ... \times_R \boldsymbol{L}_R^t \boldsymbol{X}_R \quad (3.18)$$

Equation (3.18) formalizes the relationship between $\tilde{\boldsymbol{\mathcal{X}}}$ and $\boldsymbol{\mathcal{X}}$. It is interesting to understand the role played by the terms $\{\boldsymbol{L}_r^t \boldsymbol{X}_r\}$. One recalls that both $\boldsymbol{L}_r$ and $\boldsymbol{X}_r$

define orthonormal bases; the corresponding basis vectors $\boldsymbol{l}_{i_r}^{(r)}$ and $\boldsymbol{x}_{i_r}^{(r)}$ are indeed the eigenvectors that characterize the corresponding subspaces. As a result

$$\boldsymbol{L}_r^t \boldsymbol{X}_r = \begin{bmatrix} |\boldsymbol{l}_1||\boldsymbol{x}_1| cos(\theta_{11}) & \dots & |\boldsymbol{l}_1||\boldsymbol{x}_{I_r}| cos(\theta_{1I_r}) \\ \vdots & \ddots & \vdots \\ |\boldsymbol{l}_{I_r}||\boldsymbol{x}_1| cos(\theta_{I_r1}) & \dots & |\boldsymbol{l}_{I_r}||\boldsymbol{x}_{I_r}| cos(\theta_{I_rI_r}) \end{bmatrix} \tag{3.19}$$

where θ_{ij} is the angle between the basis vector $\boldsymbol{l}_i$ and the basis vector $\boldsymbol{x}_j$. The subscript n has been omitted in the single matrix elements for the sake of conciseness and readability.

Equation (3.19) can be further revised by taking into account that -by definition- $|l_{i_r}^{(r)}| = 1$ and $|\boldsymbol{x}_{i_r}^{(r)}| = 1$. Therefore, one can conclude that the following equation holds for the mode-n unfolding of $\tilde{\boldsymbol{\mathcal{X}}}$, $\tilde{\boldsymbol{\mathcal{X}}}_{(r)}$:

$$\tilde{\boldsymbol{\mathcal{X}}}_{(r)} = \boldsymbol{L}_r^t \boldsymbol{X}_r \bar{\boldsymbol{\mathcal{X}}}_{(r)} = \begin{bmatrix} cos(\theta_{11}) & \dots & cos(\theta_{1I_r}) \\ \vdots & \ddots & \vdots \\ cos(\theta_{I_r1}) & \dots & cos(\theta_{I_rI_r}) \end{bmatrix} \cdot \bar{\boldsymbol{\mathcal{X}}}_{(r)} \tag{3.20}$$

The above equation clarifies that the discrepancies between $\tilde{\boldsymbol{\mathcal{X}}}$ and the "actual" core tensor $\bar{\boldsymbol{\mathcal{X}}}$ stem from the divergences between the bases $\boldsymbol{L}_r$ and $\boldsymbol{X}_r$. In particular, one has that

$$\boldsymbol{L}_r^t \boldsymbol{X}_r = \begin{bmatrix} cos(\theta_{11}) & \dots & cos(\theta_{1I_r}) \\ \vdots & \ddots & \vdots \\ cos(\theta_{I_r1}) & \dots & cos(\theta_{I_rI_r}) \end{bmatrix} = \boldsymbol{I} \tag{3.21}$$

only if the eigenvectors $\boldsymbol{l}_i$ and $\boldsymbol{x}_i$ are parallel for any $i = 1, .., I_r$ and if $\boldsymbol{l}_i$ is orthogonal to $\boldsymbol{x}_j$ for any couple (i, j). It is worth noting that $\tilde{\boldsymbol{\mathcal{X}}}$ encompasses the information provided by the n-mode eigenvalues of $\boldsymbol{\mathcal{X}}$ (as per (3.7)). As a consequence, one expects that $\tilde{\boldsymbol{\mathcal{X}}}$ combines this information with the relative alignment of the eigenvectors of $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$. This is an important issue because eigenvalues and eigenvectors may reveal details on geometric patterns in tensors [84]. In the following, this aspect is analyzed by providing some practical examples that help to further clarify the characteristics of the proposed similarity notion. In terms of energy realignment the pseudo core tensor $\tilde{\boldsymbol{\mathcal{X}}}$ has a simple and clear explanation. It provides all the energy contribution of the tensor $\boldsymbol{\mathcal{X}}$ along the extremal energy directions of $\boldsymbol{\mathcal{L}}$.

3.2.2 Toy Examples

In principle, the pseudo-core tensor of $\boldsymbol{\mathcal{X}}$, $\tilde{\boldsymbol{\mathcal{X}}}$, cannot be expected to inherit the properties that characterize an actual core tensor. Thus, $\tilde{\boldsymbol{\mathcal{X}}}$ usually is not an all-orthogonal, ordered tensor, nor can the Frobenius norm $\|\tilde{\boldsymbol{\mathcal{X}}}_{(r)}(i_r,:)\|$ be interpreted as a proper eigenvalue. The expression (3.20), however, suggests that the quantities $\|\tilde{\boldsymbol{\mathcal{X}}}_{(r)}(i_r,:)\|, (i_r = 1,..,I_r)$ convey useful information on the similarity between $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$ along the n-th mode. A practical example may prove useful.

First, let $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$ be gray scale images, i.e., tensors of order 2 ($\boldsymbol{\mathcal{L}}, \boldsymbol{\mathcal{X}} \in \mathbb{R}^{I_1 \times I_2}$). Then, one has

$$\boldsymbol{\mathcal{L}} = \bar{\boldsymbol{\mathcal{L}}} \times_1 \boldsymbol{L}_1 \times_2 \boldsymbol{L}_2, \tag{3.22}$$

$$\boldsymbol{\mathcal{X}} = \bar{\boldsymbol{\mathcal{X}}} \times_1 \boldsymbol{X}_1 \times_2 \boldsymbol{X}_2, \tag{3.23}$$

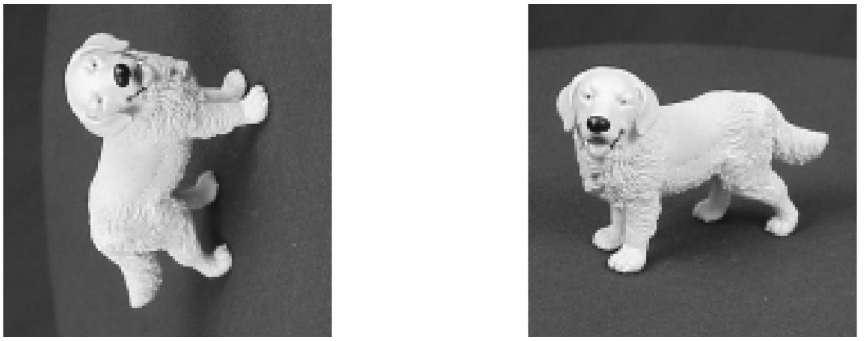
The images in Fig. 3.2(a) and 3.2(b) were used as the landmark, $\boldsymbol{\mathcal{L}}$, and the datum, $\boldsymbol{\mathcal{X}}$, respectively. In fact, the second tensor (image) was obtained by flipping vertically and rotating the first one. These operations did not affect the eigenvalues of the matrix $\boldsymbol{\mathcal{L}}$. Figures 3.2(c) and 3.2(d) show two different reconstructions of $\boldsymbol{\mathcal{X}}$. Figure 3.2(c) was obtained as

$$\boldsymbol{\mathcal{X}}' = \bar{\boldsymbol{\mathcal{X}}} \times_1 \boldsymbol{L}_1 \times_2 \boldsymbol{L}_2, \tag{3.24}$$

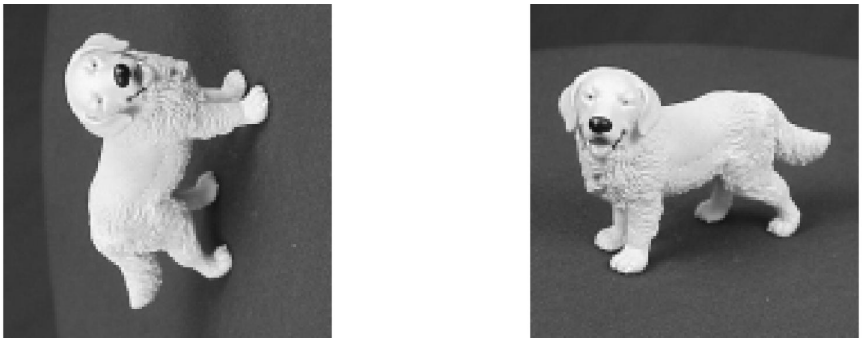
The reconstruction $\boldsymbol{\mathcal{X}}'$ exploits the original bases provided by $\boldsymbol{\mathcal{L}}$ (i.e., the eigenvectors of $\boldsymbol{\mathcal{L}}$); the eigenvalues obviously stem from $\bar{\boldsymbol{\mathcal{X}}}$. Figure 3.2 (d), instead, gives the eventual reconstruction of $\boldsymbol{\mathcal{X}}$ obtained as

$$\boldsymbol{\mathcal{X}}'' = \tilde{\boldsymbol{\mathcal{X}}} \times_1 \boldsymbol{L}_1 \times_2 \boldsymbol{L}_2, \tag{3.25}$$

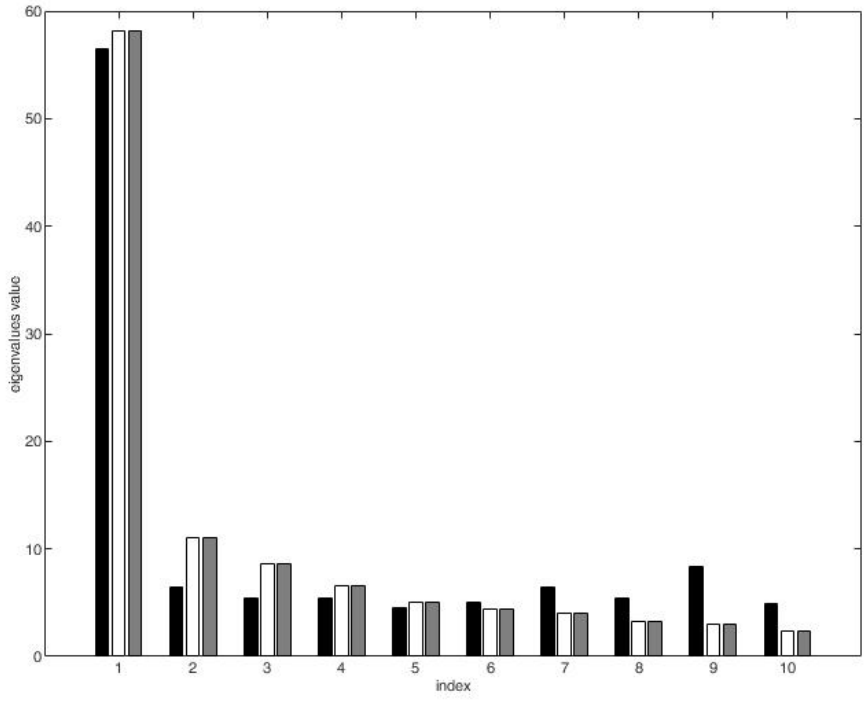
The reconstruction $\boldsymbol{\mathcal{X}}''$ still exploits the bases provided by $\boldsymbol{\mathcal{L}}$, but the eigenvalues stem from $\tilde{\boldsymbol{\mathcal{X}}}$ (in fact, they are "pseudo-eigenvalues"). Figures 3.2(c) shows that $\boldsymbol{\mathcal{X}}' \equiv \boldsymbol{\mathcal{L}}$. The reconstruction $\boldsymbol{\mathcal{X}}'$ matches $\boldsymbol{\mathcal{L}}$ since the landmark and the datum only differ in rotation. Therefore, $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$ share the same eigenvalues ($\bar{\boldsymbol{\mathcal{L}}}$ and $\tilde{\boldsymbol{\mathcal{L}}}$, respectively), although they do not share the same bases. Conversely, the reconstruction $\boldsymbol{\mathcal{X}}''$ matches $\boldsymbol{\mathcal{X}}$ (as per Fig. 3.2 (d)). This, in turn, means that $\bar{\boldsymbol{\mathcal{L}}}$ and $\tilde{\boldsymbol{\mathcal{X}}}$ do not convey the same information; i.e., they may reveal the dissimilarity between $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$.



(a) (b)



(c) (d)



(e)

FIGURE 3.2: Assessment of the degree of similarity between a landmark and a pattern according to the proposed notion of similarity, first example: (a) landmark $\boldsymbol{\mathcal{L}}$; (b) pattern $\boldsymbol{\mathcal{X}}$; (c) $\boldsymbol{\mathcal{X}}'$; (d) $\boldsymbol{\mathcal{X}}''$; (e) a plot that shows the ten largest eigenvalues of $\tilde{\boldsymbol{\mathcal{X}}}_{(1)}$ (black bars), $\bar{\boldsymbol{\mathcal{X}}}_{(1)}$ (white bars), and $\bar{\boldsymbol{\mathcal{L}}}_{(1)}$ (gray bars).

Figure 3.2(e) allows for further understanding of the role played by $\tilde{\boldsymbol{\mathcal{X}}}$ in the representation formalized in (3.22). The reconstructions $\tilde{\boldsymbol{\mathcal{X}}}'$ and $\tilde{\boldsymbol{\mathcal{X}}}''$ correspond to two different implementations of eq. (3.22), which differ in the settings of coefficients $\tilde{\sigma}_i^{\boldsymbol{X}}$.

In $\boldsymbol{\mathcal{X}}'$, one uses the values $\|\bar{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$; instead, in $\boldsymbol{\mathcal{X}}''$, the coefficients are given by $\|\tilde{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$. In the graph in Fig. 3.2(e), the x axis gives the index i, with $i = 1, .., 10$, whereas black, white, and gray bars gives the values $\|\tilde{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$, $\|\bar{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ and $\|\bar{\boldsymbol{\mathcal{L}}}_{(1)}(i,:)\|$ respectively. The values $\|\bar{\boldsymbol{\mathcal{L}}}_{(1)}(i,:)\|$ and $\|\bar{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ correspond to the 10 largest eigenvalues of $\bar{\boldsymbol{\mathcal{L}}}_{(1)}$ and $\bar{\boldsymbol{\mathcal{X}}}_{(1)}$, respectively. As expected, those values are ordered in decreasing order and are pairwise identical. On the other hand, the components $\|\tilde{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ can be regarded as some "special" eigenvalues that provide information about the process of reconstructing $\boldsymbol{\mathcal{X}}$ by using the set of basis $\boldsymbol{\mathcal{L}}_r$; in fact, the ten components are not strictly ordered from the largest to the smallest. A pairwise comparison between the values $\|\bar{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ and $\|\tilde{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ confirms that the relative weights assumed by the corresponding ten components differ. This explains the differences between $\boldsymbol{\mathcal{X}}''$ and $\boldsymbol{\mathcal{X}}'$.

Figure 3.3 presents the results of a similar experiment. In this case, $\boldsymbol{\mathcal{L}}$ (Fig. 3.3(a)) and $\boldsymbol{\mathcal{X}}$ (Fig. 3.3(b)) are two completely dissimilar images. Fig. 3.3(c) and 3.3(d) show the two different reconstruction of $\boldsymbol{\mathcal{X}}$: $\boldsymbol{\mathcal{X}}'$ and $\boldsymbol{\mathcal{X}}''$, respectively. $\boldsymbol{\mathcal{X}}'$ reveals that $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$ do not share the eigenvalues (as expected); as a result, the reconstruction somewhat mixes the landmark with the datum. On the other hand, by definition the reconstruction $\boldsymbol{\mathcal{X}}''$ matches $\boldsymbol{\mathcal{X}}$, since a realignment of the basis has been involved. Thus, yet again, it is confirmed that $\bar{\boldsymbol{\mathcal{X}}}$ and $\tilde{\boldsymbol{\mathcal{X}}}$ convey different conclusions about the degree of similarity between $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$. In this regard, Figure 3.3(e) compares the values $\|\hat{\boldsymbol{\mathcal{L}}}_{(1)}(i,:)\|$, $\|\bar{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$, $\|\tilde{\boldsymbol{\mathcal{X}}}_{(1)}(i,:)\|$ by using the same format of Fig. 3.2(e).

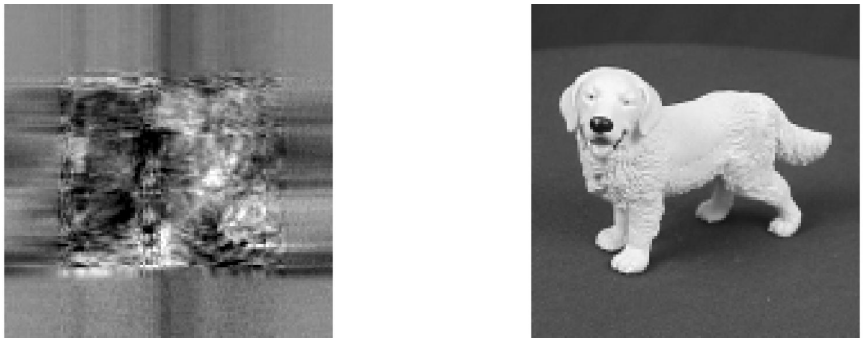
3.2.3 Proposed Similarity Function

Once that eigenvalues and pseudo eigenvalues of the couple $\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}}$ are computed, a pair of vectors collects the associate n-mode eigenvalues in $\boldsymbol{\Sigma}_{(r)}^{\boldsymbol{\mathcal{L}}} = \{\sigma_{i_r(r)}; i_r = 1, ..., I_r\}$ and $\boldsymbol{\Sigma}_{(r)}^{\boldsymbol{\mathcal{X}}} = \{\tilde{\sigma}_{i_r(r)}; i_r = 1, ..., I_r\}$. The resulting eigenvalues feed the metric procedure that works out the actual result. The goal of the proposed similarity notion is the measure of the extremal energy magnitude along the most informative directions for the landmark. With this rationale, it is sufficient a comparison between the eigenvalues and the pseudo-eigenvalues

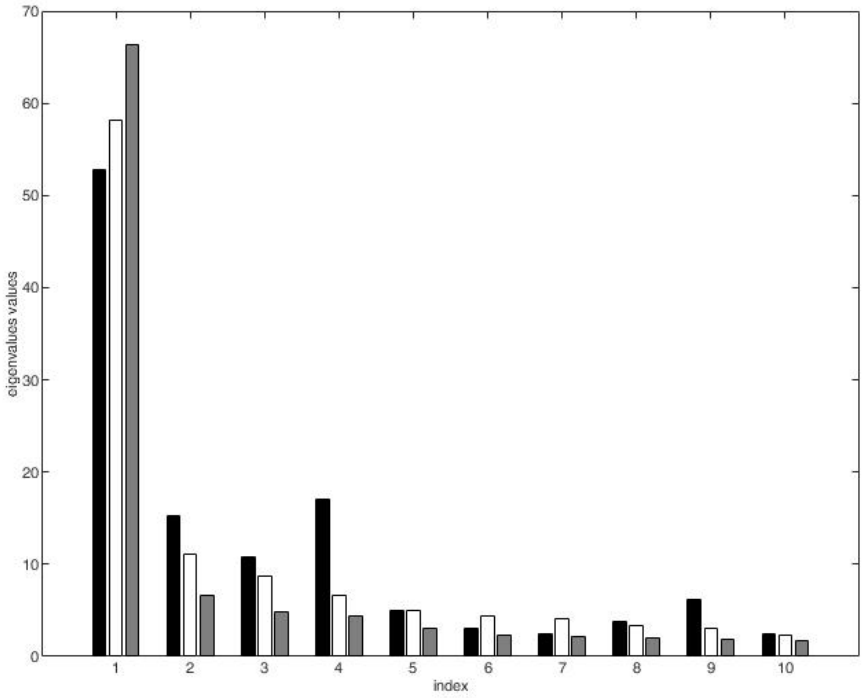
$$K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}}) = \frac{2}{1+\delta} - 1 \tag{3.26}$$

where

(a) (b)



(c) (d)



(e)

FIGURE 3.3: Assessment of the degree of similarity between a landmark and a pattern according to the proposed notion of similarity, second example: (a) landmark $\boldsymbol{\mathcal{L}}$; (b) pattern $\boldsymbol{\mathcal{X}}$; (c); $\boldsymbol{\mathcal{X}}'$ (d) $\boldsymbol{\mathcal{X}}''$ (e) a plot that shows the ten largest eigenvalues of $\tilde{\boldsymbol{\mathcal{X}}}_{(1)}$ (black bars), $\bar{\boldsymbol{\mathcal{X}}}_{(1)}$ (white bars), and $\bar{\boldsymbol{\mathcal{L}}}_{(1)}$ (gray bars).

$$\delta = \sqrt{\sum_{n=1}^{R} \sum_{i_r=1}^{I_r-\zeta_{(r)}} \frac{(\sigma_{i_r(r)} - \tilde{\sigma}_{i_r(r)})^2}{\sigma_{i_r(r)} \cdot \tilde{\sigma}_{i_r(r)}}} \tag{3.27}$$

The above metric compares the pairs $(\sigma_{i_r(r)}, \tilde{\sigma}_{i_r(r)})$, which stem from aligned basis constituted by the extremal energy directions and thus can characterize the degree of similarity between $\boldsymbol{\mathcal{L}}$ and $\boldsymbol{\mathcal{X}}$. The normalization term plays a crucial role in equation (3.27), as the eigenvalues in general can span a wide range of values. Thus, without a normalization term, one would face the risk of assessing δ by using only the 2 or 3 largest eigenvalues for each mode.

Remarkably, the proposed similarity function involves only one hyper-parameter per mode, i.e., the pruning parameter $\zeta_{(r)}$. It is worth noting that the parameters $\{\zeta_{(r)}; n = 1, .., N\}$ support a dimensionality reduction process that applies to the MLSVD-based characterizations of the pattern and the landmark. Such approach differs from the one implemented by standard dimensionality reduction process such as Latent Semantic Analysis (LSA) [85], which addresses vector spaces. In that case, SVD is used to capture the global characteristics of the whole training set. Conversely, the proposed similarity notion exploits MLSVD to obtain a suitable representation of the pattern itself (in a tensor space). This in turn means that one can better capture the specific properties of the single pattern. On the other hand, the risk of facing over fitting is higher. The pruning parameters are indeed designed to inhibit such problem.

Even if not strictly consistent from a mathematical point of view, it exists an interesting link between the reformulation of scalar product presented in section 2.18 and the proposed metric for tensor data. In fact, the re-interpreted scalar product of equation (2.18) shows that two patterns are compared on the basis of their projections on a reference set of directions $\hat{\boldsymbol{r}}$. The proposed metric, in the same way, compares the quantity of information propagated in a set of directions for each mode; in this case the set of basis is selected as the most informative ones for the reference point, the landmark, in order to maximize the quantity of information propagated. Finally, as the result of the fact that multi-modal projection result is not a scalar as in the case of eq. (2.18) but a core tensor, the information is compared based on the magnitude of information for each mode.

3.3 Proposed Algorithm for Tensor Inputs

The non-symmetric tensor-based similarity function $K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ presented in section 3.2 evaluates the degree of similarity between a pattern, $\boldsymbol{\mathcal{X}}$, and a landmark, $\boldsymbol{\mathcal{L}}$, by actually

processing $\tilde{\boldsymbol{\mathcal{X}}}$ and $\bar{\boldsymbol{\mathcal{L}}}$. By definition, any mode-n unfolding of $\bar{\boldsymbol{\mathcal{L}}}$ encloses the n-mode eigenvalues of $\boldsymbol{\mathcal{L}}$(3.6); at the same time, Sec. 3.2 proved that it is convenient to interpret the Frobenius norm $\|\tilde{\boldsymbol{\mathcal{X}}}_{(r)}(i_r,:)\|$ as an i_r-th r-mode eigenvalue. As a result, the two sets of eigenvalues provide the inputs for a specific measure of similarity between $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{L}}$. In this thesis the specific measure is the one introduced in equation (3.27). As a result, merging these two steps it is possible to define a notion $K(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}}) : \mathbb{R}^{I_1\times I_2\times\ldots\times I_R}\times\mathbb{R}^{I_1\times I_2\times\ldots\times I_R}\rightarrow[-1;1]$. The complete procedure is summarized in Algorithm 6.

The Algorithm 6 outlines the computation of $K(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}})$. After working out $\bar{\boldsymbol{\mathcal{L}}}$ and $\tilde{\boldsymbol{\mathcal{X}}}$ as per steps 0 and 1, step 2 collects the associate r-mode eigenvalues in $\Sigma^{\boldsymbol{\mathcal{L}}}_{(r)}=\{\sigma_{i_r(r)};i_r=1,...,I_r\}$ and $\Sigma^{\boldsymbol{\mathcal{X}}}_{(r)}=\{\tilde{\sigma}_{i_r(r)};i_r=1,...,I_r\}$. A pruning procedure at step 3 discards the least $\zeta_{(r)}$ informative eigenvalues. To ensure flexibility, an adaptive set size, $\zeta_{(r)}$, should be set for each mode, r. In step 4, the resulting eigenvalues feed the metric procedure that works out the actual result. The goal of the proposed similarity notion is the measure of the extremal energy magnitude along the directions of the landmark. Whit this rationale, it is sufficient a comparison between the eigenvalues and the pseudo-eigenvalues.

In the end, the proposed metric can be integrate in Algorithm 2 extending the paradigm based on similarity functions to data in tensor format providing a straightforward procedure for the training of predictors in tensor input domain.

3.3.1 Computational Cost

The above framework exhibits a common schema with the kernel-based framework for tensorial data introduced in [81]. The latter approach virtually extends any kernel machine to a tensor-based kernel machine. This goal is achieved by designing a suitable kernel that can exploit the algebraic structure of tensors. The decision function can then be learned by solving a single convex optimization problem. The proposed framework, in fact, exploits similarity functions to replace kernels, and the decision function can be learned by solving a single RLS problem.

A comparison of the relative computational complexities reveals the basic difference between the two approaches. In [81], the kernel function $\varphi(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}})$ that processes two generic tensors $\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}}$ is formulated as

$$\varphi(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}})=\prod_{n=1}^{R}\varphi^{(r)}(\boldsymbol{\mathcal{X}},\boldsymbol{\mathcal{L}}) \tag{3.28}$$

where

Algorithm 6 Computation of the tensor-based similarity function

Input

- a multiway pattern $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{I1 \times I_2 \times ... \times I_R}$
- a multiway landmark $\boldsymbol{\mathcal{L}} \in \mathbb{R}^{I1 \times I_2 \times ... \times I_R}$
- pruning parameters $\zeta_{(r)} (r = 1, ..., R)$

0. **Initialize**

decompose $\boldsymbol{\mathcal{L}}$ to obtain

- a core tensor $\bar{\boldsymbol{\mathcal{L}}}$
- the orthonormal bases $\boldsymbol{L}_r (r = 1, ..., R)$

1. **Psudo-tensor**

compute the pseudo-tensor of $\boldsymbol{\mathcal{X}}$

$\tilde{\boldsymbol{\mathcal{X}}} = \boldsymbol{\mathcal{X}} \times_1 \boldsymbol{L}_1^t \times_2 \boldsymbol{L}_2^t ... \times_R \boldsymbol{L}_R^t$

2. **Eigenvalues**

1. extract the n-mode eigenvalues from $\bar{\boldsymbol{\mathcal{L}}} (r = 1, ..., R)$

$$\Sigma_{(r)}^{\boldsymbol{\mathcal{L}}} = \{\sigma_{i_r(r)}; i_r = 1, ..., I_r\}, \ \texttt{where} \ \sigma_{i_r(r)} = \|\bar{\mathcal{L}}_{(r)}(i_r, :)\|$$

2. extract the n-mode eigenvalues from $\tilde{\mathcal{X}} (r = 1, ..., R)$

$$\Sigma_{(r)}^{\mathcal{X}} = \{\tilde{\sigma}_{i_r(r)}; i_r = 1, ..., I_r\}, \ \texttt{where} \ \tilde{\sigma}_{i_r(r)} = \|\tilde{\mathcal{X}}_{(r)}(i_r, :)\|$$

3. **Pruning**

for each mode, drop the last $\zeta_{(r)}$ eigenvalues (with $\zeta_{(r)} < I_r$)

$$\Sigma_{(r)}^{\boldsymbol{\mathcal{L}}} = \{\sigma_{i_r(r)}; i_r = 1, ..., I_r - \zeta_{(r)}\}, \ \ \Sigma_{(r)}^{\boldsymbol{\mathcal{X}}} = \{\sigma_{i_r(r)}; i_r = 1, ..., I_r - \zeta_{(r)}\}$$

4. **Similarity**

compute the similarity

$K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}}) = \frac{2}{1+\delta} - 1$

where

$\delta = \sqrt{\sum_{r=1}^{R} \sum_{i_r=1}^{I_r - \zeta_{(r)}} \frac{(\sigma_{i_r(r)} - \tilde{\sigma}_{i_r(r)})^2}{\sigma_{i_r(r)} \cdot \tilde{\sigma}_{i_r(r)}}}$

$$\varphi^{(r)}(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}}) = exp\Big(- \frac{1}{\sigma^2}(I_r - trace(\boldsymbol{Z}^t \boldsymbol{Z}))\Big), \tag{3.29}$$

$$\boldsymbol{Z} = (\hat{\boldsymbol{V}}^{\boldsymbol{\mathcal{X}}}_{(r)})^t (\hat{\boldsymbol{V}}^{\boldsymbol{\mathcal{L}}}_{(r)}), \tag{3.30}$$

In the expression (3.30), $\hat{\boldsymbol{V}}^{\boldsymbol{\mathcal{X}}}_{(r)}$ is the matrix computed by applying SVD to $\boldsymbol{\mathcal{X}}_{(r)}$ as per equation (3.3), and is obtained by picking the first r columns of $\hat{\boldsymbol{V}}^{\boldsymbol{\mathcal{X}}}_{(r)}$, with $r = \texttt{rank}(\boldsymbol{\mathcal{X}}_{(r)})$. A similar procedure applies to $\hat{\boldsymbol{V}}^{\boldsymbol{\mathcal{L}}}_{(r)}$. Thus the computation of $\varphi(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ requires a set of $2R$ SVD's (as both $\boldsymbol{\mathcal{X}}$ and $\boldsymbol{\mathcal{L}}$ are processed). Then, two matrix products for each $\varphi(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ must be performed. Therefore, the cost $\boldsymbol{O}_{KF}$ associated to the computation of $\varphi(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ is

$$\boldsymbol{O}_{KF} \cong 2 \cdot R \cdot \boldsymbol{O}_{SVD} + 2 \cdot R \cdot \boldsymbol{O}_{MP} \tag{3.31}$$

For the training of the overall kernel machine, one uses $\varphi(\bullet, \bullet)$ to compute the symmetric kernel matrix. Thus the computational cost $\boldsymbol{O}_{TGK}$ associated to the kernel building can be estimated as

$$\boldsymbol{O}_{TGK} \cong Z \cdot R \cdot \boldsymbol{O}_{SVD} + R \cdot Z^2 \cdot \boldsymbol{O}_{MP} \tag{3.32}$$

The implementation of the kernel-based decision function requires a set of Z inner products $\varphi(\bullet, \bullet)$, involving the test pattern versus all training patterns. On one hand, the MLSVD result of each training pattern is computed offline and is stored in memory; however, the MLSVD of the test pattern has to be worked out online. Thus, the computational cost $\boldsymbol{O}_{TTK}$ associated to such step is:

$$\boldsymbol{O}_{TTK} \cong R \cdot \boldsymbol{O}_{SVD} + 2 \cdot R \cdot Z \cdot \boldsymbol{O}_{MP} \tag{3.33}$$

In the proposed framework, the computational complexity associated to the similarity function $K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ can be estimated easily (as per Algorithm 6). One first needs a set of R SVD procedures to complete Step 0. Then, R matrix products are required to compute $\tilde{\boldsymbol{\mathcal{X}}}$ (Step 1). The computations involved by subsequent steps are negligible in terms of computational complexity as compared with Step 0 and Step 1. The overall cost, $\boldsymbol{O}_{SF}$, associated with the computation of $K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ is:

$$\boldsymbol{O}_{SF} \cong R \cdot \boldsymbol{O}_{SVD} + R \cdot \boldsymbol{O}_{MP} \tag{3.34}$$

One might argue that associating a uniform cost, $\boldsymbol{O}_{SVD}$, to any SVD is incorrect, since each unfolding of $\boldsymbol{\mathcal{L}}$ has its own size, which in turn affects the subsequent SVD. The same consideration applies to $\boldsymbol{O}_{MP}$. This issue, however, only gets relevant when one wants to formalize the relationship between $\boldsymbol{O}_{SF}$ and $(I_1, I_2, .., I_r)$.

To address the training of eventual predictor (2.11), one applies $K(\boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{L}})$ to complete the mapping of the input patterns in the remapped space $\mathbb{R}^N$. Thus, as per Step 1 of Algorithm 2, one completes a set of N MLSVDs (one for each landmark). Then, the Z input patterns are finally remapped. In summary the cost of the prediction training is:

$$\boldsymbol{O}_{TGS} \cong N \cdot R \cdot \boldsymbol{O}_{SVD} + N \cdot Z \cdot R \cdot \boldsymbol{O}_{MP} \tag{3.35}$$

This means that $\boldsymbol{O}_{TGS} = \boldsymbol{O}_{TGK}$ when, in Algorithm 2, one sets all training patterns as landmarks (i.e., $N = Z$).

After training is completed, the implementation of the decision function (2.11) only requires the availability of the landmark bases $\boldsymbol{L}_r$, which are stored in memory. This is a major difference with respect to the decision function proposed in [75]. Thus, when a test pattern is classified, the computational cost associated to the mapping stage is

$$\boldsymbol{O}_{TTS} \cong N \cdot R \cdot \boldsymbol{O}_{MP}. \tag{3.36}$$

Clearly, $\boldsymbol{O}_{TTS}$ can prove significantly lower than $\boldsymbol{O}_{TTK}$, which is made heavier by the need of computing the MLSVD of the test pattern [34]. This aspect becomes crucial when targeting the implementation of the predictor on a digital architecture, since the SVD can prove quite demanding in terms of computational complexity.

3.4 Comparison with Existing Literature

The proposed framework exhibits distinctive features as compared with existing approaches to tensor-based learning. First, the framework differs from conventional approaches that complete learning by an iterative process [75–78]. In general, these approaches solve a convex optimization problem at each step; hence, the learning processes are characterized by a huge computational cost. Nonetheless, one should deal with the setup of free parameters related to the convergence criterion.

SHTM [79] reformulates the iterative approach implemented by Supervised Tensor Learning (STL) [76] to the purpose of obtaining a framework that inherits the linear C-SVM

format [79]; as a result, the eventual learning process involves a single, standard convex optimization problem. The SHTM framework exploits CANDECOMP/PARAFAC (CP) [86] decomposition to suitably implement the inner product between tensors, which are replaced by their rank-one decomposition. Thus, a comparison between SHTM and the proposed framework reveals two basic differences: first, SHTM uses tensor decomposition specifically to address the inner product computation in a SVM-based learning scheme; secondly, SHTM yields a predictor that requires a (computationally demanding) decomposition also for classifying a test pattern at run time. Tensor decomposition also plays a role in projection learning methods [71, 72, 74], which aim at learning bases that can suitably capture manifolds in a training set. Most of such approaches convey a minimization problem over the entire training set. The proposed framework, instead, by applying MLSVD in conjunction with landmarks attains two goals: the manifold proprieties of the training corpus are characterized, and the peculiar properties of each pattern are preserved. Algorithm 6 tackles the risk of overfitting by a pruning mechanism on the eigenvectors (as per Step 3).

Convolutional neural networks (CNNs) [87] are indeed designed to deal with n-th order tensors by exploiting convolutional operations. The main advantage of CNNs with respect to shallow architectures is the hierarchical organization of the information supported by the multi-layer outline. This feature proves very effective in domains where patterns embeds structured information, e.g., image processing and video processing. On the other hand, CNNs are computationally demanding and requires very large datasets for the training phase. Thus, shallow architectures represent the only viable option when one targets implementations on resource-constrained devices.

3.5 Experimental Results

The experimental verification of the proposed paradigm aimed at evaluating the accuracy of the tensor-based learning scheme derived from the theory of similarity function. The overall framework was tested on seven classification problems involving multiway data. The first problem used a synthetic dataset that had already been used in [81] for estimating the performance of the tensor kernel (3.28). The other six problems addressed real-world domains, namely, ETH-80 [88], Yale Faces [89], Flower [90], KTH [91], Cambridge Hand Gesture [92], and Gas Sensors [93]. The proposed datasets covered a wide range of applications. As a major result of the datasets diversity the comparison between the proposed approach and state-of-the-art methodologies became robust against the peculiarities of the single applications.

The experiments were all designed to assess the generalization performance of the proposed tensor-based classifier. At the same time, the tests supported comparisons with three alternative solutions: SHTM [79]; a Support Vector Machine (SVM) that processed multiway data by relying on the tensor kernel (3.28); a standard SVM that first remapped multiway data into vector representations and then adopted the conventional RBF kernel. The first two solutions refer to state-of-the-art approaches that address tensor-based learning by involving a single, standard convex optimization problem. The latter solution allows one to understand the performance of a conventional approach to the problem.

In each experiment, the proposed framework, SHTM, the tensor-based SVM, and the basic SVM were compared by defining a common set up for the range of admissible values of:

- λ, i.e., the regularization parameter ($C = 1/\lambda$ for SVM's);
- σ, i.e., the hyper-parameter of both the standard RBF kernel and the tensor kernel (3.28).
- R, i.e., the rank parameter of SHTM.

The following settings were adopted:

- $\lambda \in \{1 \cdot 10^{-6}, 1 \cdot 10^{-5}, 1 \cdot 10^{-4}, 1 \cdot 10^{-3}, 1 \cdot 10^{-2}, 1 \cdot 10^{-1}, 1, 1 \cdot 10^{1}, 1 \cdot 10^{2}, 1 \cdot 10^{3}, 1 \cdot 10^{4}, 1 \cdot 10^{5}, 1 \cdot 10^{6}\}$
- $\sigma \in \{1 \cdot 10^{-6}, 1 \cdot 10^{-5}, 1 \cdot 10^{-4}, 1 \cdot 10^{-3}, 1 \cdot 10^{-2}, 1 \cdot 10^{-1}, 1, 1 \cdot 10^{1}, 1 \cdot 10^{2}, 1 \cdot 10^{3}, 1 \cdot 10^{4}, 1 \cdot 10^{5}, 1 \cdot 10^{6}\}$,
- $R \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$,

Conventional model-selection procedures always drove the run-time settings of $\{\lambda, \sigma\}$ (or $\{\lambda, R\}$) for evaluating generalization performance on unseen test data. In the case of the proposed framework, model selection also supported the setting of the pruning parameters, $\zeta_{(r)}$, used in Algorithm 6; in addition, all the patterns included in the training set served as landmarks (i.e., $L = Z$).

The following sections present the results of the experimental sessions, and address the performance of the four different solutions on each dataset.

All the simulation were performed using Matlab software.

Synthetic Dataset: Classification of Sparsity Patterns

The first domain addressed the classification of third-order tensors characterized by two different types of sparsity patterns. This synthetic dataset was used in [81] for demonstrating the generalization performance of a SVM adopting the tensor kernel (3.28).

The patterns were generated as follows. Let $\boldsymbol{e}_j \in \mathbb{R}^I$ denote the j-th canonical basis vector with $e_{ij} = 1$ if $i = j$ and $e_{ij} = 0$ otherwise. In addition, let $\boldsymbol{\mathcal{D}}_j \in \mathbb{R}^{I \times I \times I}$ be the rank-1 tensor defined as: $\boldsymbol{\mathcal{D}}_j = e_j \otimes e_j \otimes e_j$. Then, the m-th pattern, $\boldsymbol{\mathcal{X}}_m$, was computed as

$$\boldsymbol{\mathcal{X}}_m = \begin{cases} a_m \boldsymbol{\mathcal{D}}_1 + b_m \boldsymbol{\mathcal{D}}_2 + c_m \boldsymbol{\mathcal{D}}_3 + \boldsymbol{\mathcal{G}}_m, & \text{if } y_m = 1. \\ a_m \boldsymbol{\mathcal{D}}_4 + b_m \boldsymbol{\mathcal{D}}_5 + c_m \boldsymbol{\mathcal{D}}_6 + \boldsymbol{\mathcal{G}}_m, & \text{if } y_m = -1. \end{cases} \tag{3.37}$$

In equation (3.37), a_m, b_m and c_m were i.i.d. from a zero-mean Gaussian distribution with variance $1 - \beta^2$; $\boldsymbol{\mathcal{G}}_m$ was a noise tensor, whose entries were i.i.d. from a zero-mean Gaussian distribution with variance β^2.

The experimental dataset was generated by adopting $I = 3$ and $\beta^2 = 0.05$. Each experiment conveyed a binary classification problem with a balanced training set and a balanced test set. The size of the training set took on the values $\{10, 20, 50, 100, 200\}$; the test set always included 100 samples. For each size of the training set, ten different runs were completed, requiring ten different pairs of training/test sets.

Table 3.1 gives the experimental outcomes obtained: rows relate to the size of the training set; columns refer to the four classifiers. The performance was assessed by averaging the classification error (expressed in the range $[0, 1]$) over the ten runs; the associate standard deviation is given in brackets. Empirical results point out that the proposed framework outperformed both SHTM and the standard SVM. The SVM based on the tensor kernel only outperformed the proposed framework when very limited training sets were involved.

TABLE 3.1: Synthetic Dataset: Results of the Experimental Session: the four algorithms are compared in term of average accuracy (%) paired with standard deviation

Size	Proposed Framework	SHTM	Multiway SVM	Standard SVM
10	0.005 (0.009)	0.3275 (0.087)	**0.000** (0.002)	0.172 (0.085)
20	0.003 (0.004)	0.3285 (0.058)	**0.000** (0.002)	0.081 (0.056)
50	**0.000** (0.001)	0.3185 (0.057)	0.000 (0.002)	0.023 (0.019)
100	**0.000** (0.000)	0.3595 (0.067)	**0.000** (0.000)	0.012 (0.010)
200	**0.000** (0.000)	0.3605 (0.063)	**0.000** (0.000)	0.006 (0.007)

ETH-80

The ETH-80 database [88] contains images of objects grouped into 8 categories (apples, pears, tomatoes, cars, cups, cows, horses, dogs). Each category covers 10 objects that span large in-class variations, while still clearly belonging to the category. Each object is represented by 41-color images, which correspond to as many viewpoints equally spaced over the upper viewing hemisphere. All images are sized 128×128 pixels. Accordingly, each of them is represented as a third-order tensor $\mathbb{R}^{128 \times 128 \times 3}$.

The session was designed to evaluate the performance on the several pairwise binary problems that stem from this dataset. Each experiment targeted the classification problem "category A" versus "category B", hence a total of 28 tests were completed to cover all combinations.

In each experiment, six objects were randomly drawn and included in the training set (three for each category), thus the training set held $6 \times 41 = 246$ patterns. The test set included one randomly extracted object per class, for a total of $2 \times 41 = 82$ test patterns. The training set and the test set never shared any pattern. Ten runs per experiment were completed, requiring ten different pairs of training/test sets.

Figure 3.4 shows the results of the overall set of experiments. In the graph, the x axis marks each binary classification problem, whereas the y axis gives the average classification error over the ten runs (expressed in the range $[0, 1]$). For the sake of clarity, the graph plots -for each single experiment- two quantities: the classification error scored by the proposed framework (black thick line) and the best classification error achieved among the three remaining frameworks (dashed gray line). The graph points out that the proposed approach was never outperformed by state-of-the-art approaches. In several cases, the classification error scored by the proposed approach was significantly smaller. To amplify on that, Table 3.2 provides a detailed analysis of these results, aimed at assessing the statistical significance of the results from each experiment. In a given test, predictor A was considered "better than" predictor B only if

$$\bar{\mu}_A + \bar{\sigma}_A/\sqrt{10} < \bar{\mu}_B - \bar{\sigma}_B/\sqrt{10} \tag{3.38}$$

where $\bar{\mu}$ and $\bar{\sigma}$ are the sample mean and the sample standard deviation, respectively, worked out on the ten classification errors. The expression (3.38) takes into account the standard error in the computation of the average classification error (i.e., the sample mean). The columns of Table 3.2 give the following quantities:

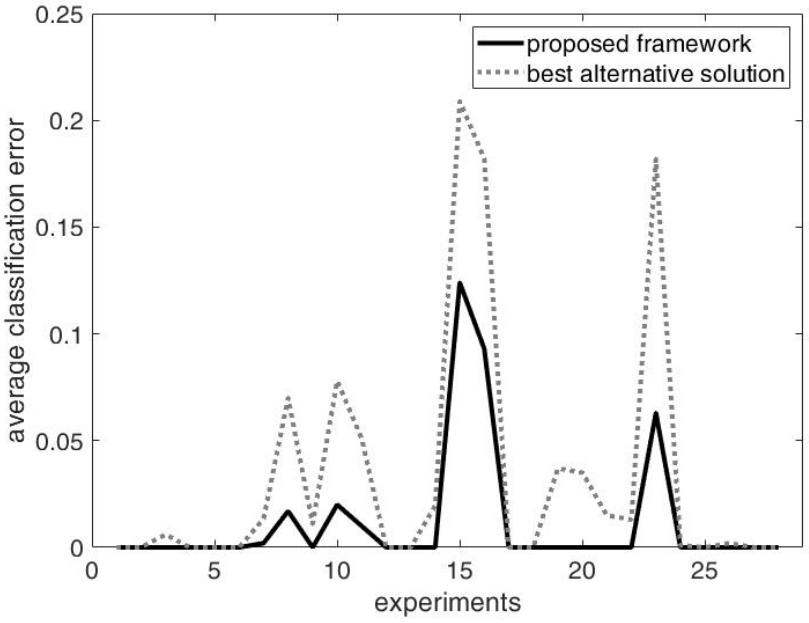


FIGURE 3.4: ETH-80 dataset: results of the 28 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

- ν_B: the number of experiments in which the proposed framework proved to be the best predictor, as per (3.38);
- ν_{BNE}: the number of experiments in which the proposed framework still scored the lowest classification error, but equation (3.38) did not hold;
- ν_{BH}: the number of experiments in which the proposed framework proved better than SHTM, as per (3.38);
- ν_{BT}: the number of experiments in which the proposed framework proved better than tensor kernel SVM, as per (3.38);
- ν_{BS}: the number of experiments in which the proposed framework proved better than standard SVM, as per (3.38);
- μ_G: the average improvement in classification error attained by the proposed framework. This quantity was computed by considering only the ν_B experiments in which the method proved to be the best predictor. The gain always referred to the second-best comparison.

The table also provides -in brackets- the same quantities expressed as percentage over the total number of experiment. The results confirm the effectiveness of kernel-based learning with similarity functions, which in almost half of the experiments scored as the best overall predictor; the average gain in classification error was 0.05.

Yale Faces

The Yale Face Database [89] contains 165 grayscale images of 15 individuals. There are 11 images per subject, one per different facial expression or configuration. All images have size 243×320 pixels; the present experimental session, tough, exploited the processed data set already used in [94]. Accordingly, each image was represented as a second-order tensor $\mathbb{R}^{64\times64}$.

The experimental session addressed the binary classification problems "subject A" versus "subject B" (105 experiments in total). In each experiment, the training set was generated by randomly drawing 9 images per class; the test set included the remaining 2 images per class. The training set and the test set never shared any pattern. A total of ten different runs per experiment were completed, hence ten different training/test pairs of sets were generated. Figure 3.5 gives the results of the overall set of experiments and uses the same the format as per Fig. 3.4. The x axis marks single experiments; the y axis gives the classification error on the test sets. The plot shows that the proposed approach was never outperformed by state-of-the-art approaches. Indeed, Table 3.3 provides further details on the experimental outcomes by using the same descriptors as per Table 3.2. The proposed framework often scored the lowest classification error; nonetheless, frequently the second-best predictor was very close to such performance (as per BNE). This result may indicate that the Yale Faces Database does not provide a very challenging problem in general.

Flower

The flower database [90] gathers images for 17 different categories of flowers; 80 color images per category are provided. The images were characterized by changes in scale, pose and light variations; in addition, a single class may include images with a single flower in the foreground, as well as images with groups of flowers of the same class. In the experimental session, the database was pruned by only keeping those images having one flower in the foreground. As a result, 5 categories had to be removed from the

Table 3.2: ETH-80 Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
12(42.8%)	16(57.1%)	16(57.1%)	17(60.7%)	15(53.5%)	0.05

Table 3.3: Yale Faces Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
15(14.3%)	81(77.1%)	22(20.9%)	105(100%)	25(23.8%)	0.06

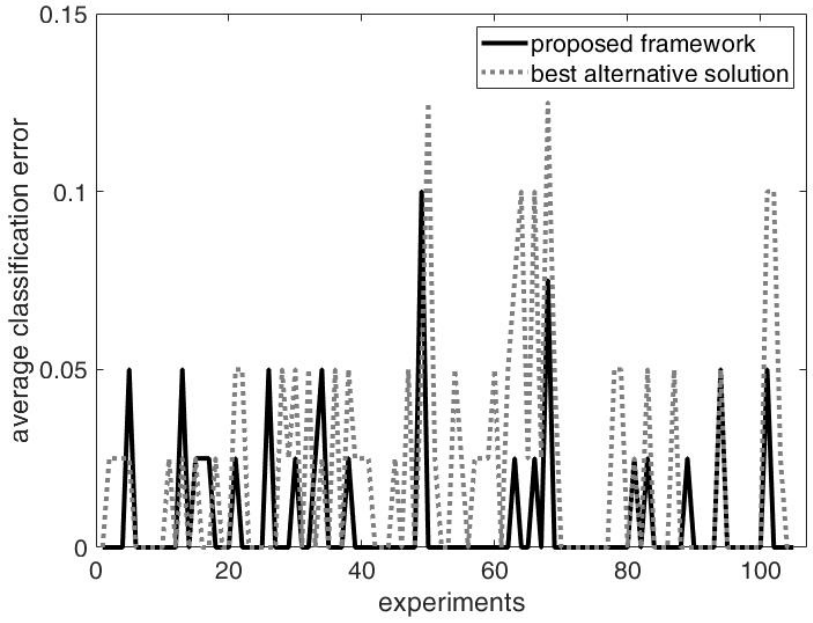


FIGURE 3.5: Yale Faces dataset: results of the 105 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

database, since the number of acceptable pictures was too small. All the pictures were resized to obtain a unique format (150×150 pixels). Thus each pattern was represented as a third-order tensor $\mathbb{R}^{150\times150\times3}$.

The experimental session evaluated the performances on the binary classification problems "category A" versus "category B" (66 experiments in total). In each test, the training set was generated by randomly drawing 20 patterns per class; the test set included 10 random patterns. The training set and the test set never shared any pattern. A total of ten different runs per experiment were completed, hence ten different training/test pairs of sets were generated.

Figure 3.6 gives the results of the overall set of experiments with the same format used in Fig. 3.4. The graph points out that the proposed framework often compared favorably with state-of-the-art frameworks. Table 3.4 provides a deeper statistical analysis of the experimental outcomes with the usual descriptors. The table confirms the effectiveness of proposed framework, which scored as the best overall predictor in more than half of the experiments. In addition, the average gain classification error was 0.06.

TABLE 3.4: Flower Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
36(55.4%)	22(33.8%)	47(72.3%)	58(89.2%)	46(70.8%)	0.06

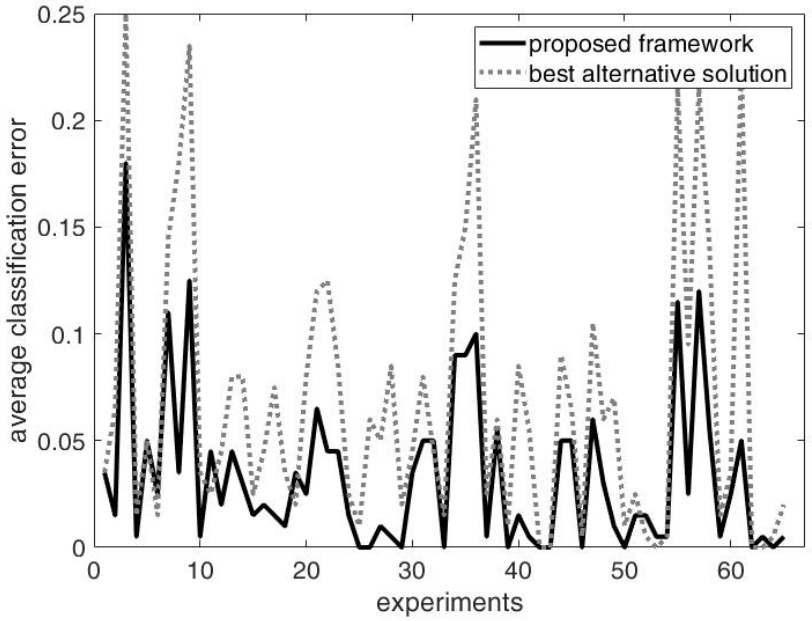


FIGURE 3.6: Flower dataset: results of the 66 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

KTH

The KTH database [91] includes six types of human actions { walking, jogging, running, boxing, hand waving and hand clapping} performed by 25 subjects. Four different scenarios were involved: outdoor, outdoor with scale variation, outdoor with different clothes, indoor. All sequences were taken over homogeneous backgrounds with a static camera (frame rate: 25fps). In the original database, the frame size was 160×120 pixels; the length of the videos varied. In the present experiment, the videos were all resized to 30×30 pixels. Accordingly, each pattern was represented as a third-order tensor $\mathbb{R}^{30 \times 30 \times I}$, with I varying according to each video.

The experimental session evaluated the performance of the proposed framework on the binary classification problems "action A" versus "action B" (15 experiments in total). In each experiment, the training set was generated by randomly drawing 30 patterns per class; the test set included 10 random patterns per class. The training set and the test set never shared any pattern. I was always set according to the shortest sequence included in the experiment (training and test); all the sequences originally including a number of frames greater than I were edited by removing all frames after the I-th one. Ten different runs per experiment were completed; hence, ten different pairs of training/test sets were generated.

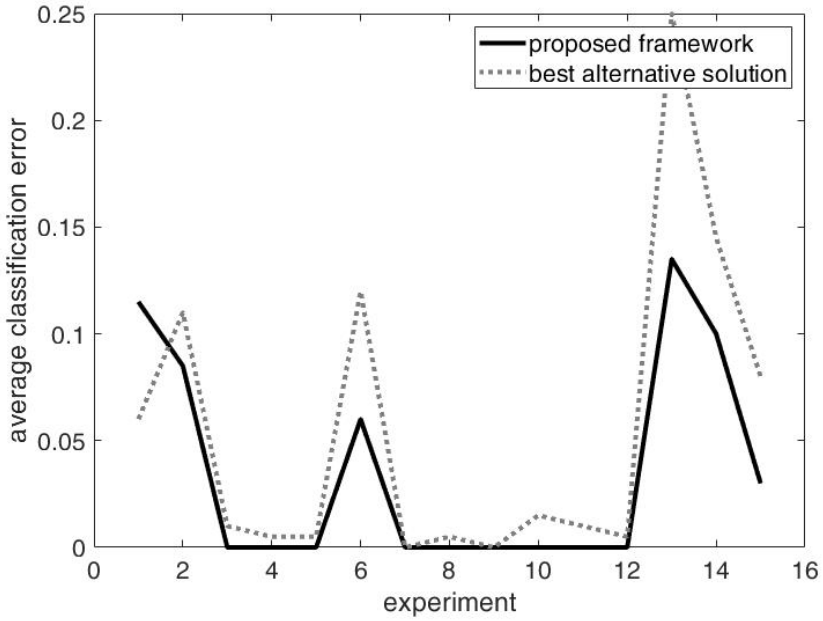


FIGURE 3.7: KTH dataset: results of the 15 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

TABLE 3.5: KTH Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
7(46.7%)	7(46.7%)	14(93.3%)	7(46.7%)	15(100%)	0.04

Figure 3.7 gives the results for the overall set of experiments with the same format used in Fig. 3.4. The graph confirms the reliability of the proposed framework. Table 3.5 gives further statistical parameters, and points out that the proposed framework scored the lowest classification error in 14 experiments out of 15 (best predictor in 7 experiments out of 15 if one only considers B); the average improvement in classification error was 0.04. Table 3.5 indeed shows that only the tensor-based SVM did proved better than the proposed approach in a few tests. Conversely, both SHTM and the basic SVM never attained the performances scored by the proposed approach.

Cambridge Hand Gesture

The Cambridge Hand Gesture database [92] includes 900 image sequences that stemmed from 9 gesture classes. The 9 gestures were defined by 3 primitive hand shapes and 3 primitive motions. Each class contained 100 image sequences, which had been recorded by involving 2 different subjects, 10 arbitrary motions and 5 different illuminations. Each sequence was recorded in front of a fixed camera having roughly isolated gestures in space and time. In the present experiment, images were all resized to the uniform

TABLE 3.6: Hand Gesture Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
3(8.3%)	15(41.7%)	35(97.2%)	3(8.3%)	34(94.4%)	0.04

format 40×40 pixels. As a result, each image sequence was represented as a third-order tensor $\mathbb{R}^{40 \times 40 \times I}$. Actually, I varied for each sequence.

The session followed the approach commonly applied in the literature for this dataset, and evaluated the performance of the proposed framework on the binary classification problems "gesture A" versus "gesture B" (36 experiments in total). For each experiment, a training set was generated by including all the image sequences captured with the plain illumination setting (20 patterns per class). The corresponding test set included all the image sequences captured with the remaining illuminations (80 patterns per class). In each experiment, I was set according to the shortest sequence included in the dataset (training and test). As a result, all the sequences originally including a number of images greater than I were subsampled.

Figure 3.8 gives the results of the overall set of experiments and uses the same the format as per Fig. 3.4. The x axis marks single experiments; the y axis gives the classification error on the test sets. The results show that, in this case, the proposed framework gained the role of best predictor only in few experiments. Table 3.6 provides further insights on the experimental outcomes. In the present setup, however, the standard error associated to the classification error was null, since experiments did not involve multiple runs. The quantity B just counted the experiments in which the proposed framework scored the lowest classification error; the quantity BNE counted the cases in which the proposed framework proved to the best predictor as well as another predictor (featuring the same classification error). The table proves that, again, only the tensor-based SVM was able to compete with the proposed framework.

Gas Sensors

The Gas Sensors database [93] includes 13,910 measurements from 16 chemical sensors exposed to 6 gases at different concentration levels. The single measurement provided the response of the sensors when exposed to a given gas at a given concentration; 8 features per sensor were used. The measurements were gathered during 36 months. In the present experimental session, the data stemming from a single measurement were organized as a second-order tensor $\mathbb{R}^{16 \times 8}$ (i.e., the 16 sensors with the corresponding 8 features). Experiments addressed the binary classification problem "gas A" versus "gas B" (16 experiments in total). Thus, concentration levels were not involved in the

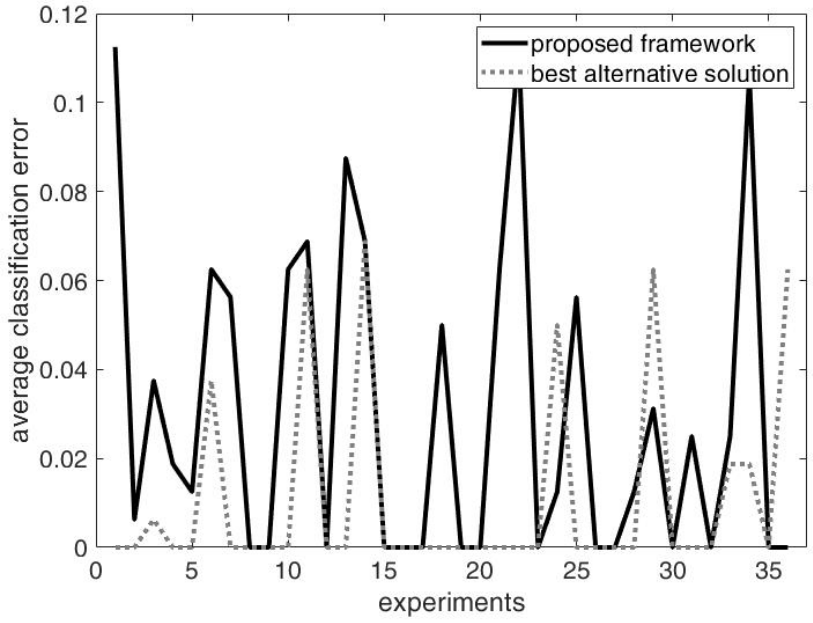


FIGURE 3.8: Hand Gesture dataset: results of the 36 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

TABLE 3.7: Gas Sensor Dataset: Analysis of the Results of the Experimental Session

ν_B	ν_{BNE}	ν_{BH}	ν_{BT}	ν_{BS}	μ_G
0(0.0%)	4(26.7%)	7(46.7%)	0(0.0%)	2(13.3%)	0.0

experiment. The balanced training set always included 25 samples per class; the test set included 100 samples per class. Ten runs were performed; in each run, the training set and the test set were generated by randomly sampling the available dataset. The training set and the test set never shared any pattern.

Figure 3.9 gives the results of the overall set of experiments and uses the same the format as per Fig. 3.4. The x axis marks single experiments; the y axis gives the classification error on the test sets. The results show that, in general, the proposed framework achieved the performance scored by the best solution among the three competitors. In fact, only in a few cases one of the competitors was able to slightly outperform the proposed framework. Table 3.7 gives further statistical parameters, and confirms that the proposed framework never proved to be the best overall predictor with a statistical significance.

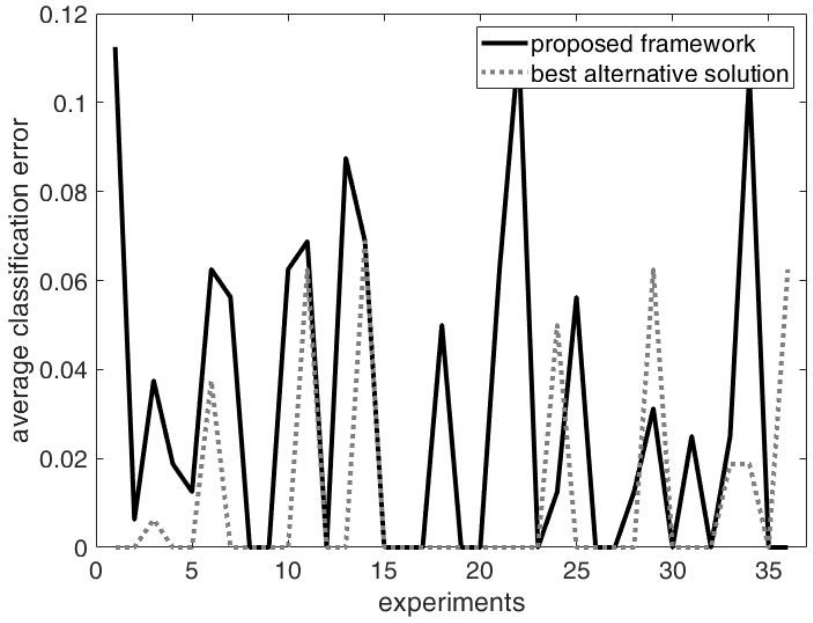


FIGURE 3.9: Gas Sensor dataset: results of the 16 experiments on binary classification. The graph compares, for each experiment, the proposed framework with the best solution achieved among state-of-the-art frameworks.

3.5.1 Concluding Remarks

This research investigated how the theory of learning with similarity functions can support the development of an efficient framework that deals with multiway data inherently. The design of an original, effective notion of similarity between tensors indeed represents the core feature of the proposed research.

The notion of similarity is built on the decomposition of a tensor into two components: the core tensor and the corresponding orthonormal basis. The degree of similarity between a pattern and a landmark is then assessed by taking into due account the alignment between the respective basis. This, in turn, means that similarity is not just estimated by conventionally comparing the standard n-mode eigenvalues of the two tensors; this procedure in fact can only provide partial information on the degree of resemblance between the tensors. Nonetheless, it is worth to note that the proposed framework can utilize MLSVD to characterize the intrinsic structural properties of each single landmark; as a major result, one can more reliably capture the underlying domain distribution.

Experimental results confirm the effectiveness of the proposed framework, which compared favorably with both a basic SVM and two state-of-the-art frameworks that can inherently process tensors: SHTM and the tensor-based SVM classifier model presented in [81]. Only the latter predictor proved able to compete with the proposed framework

in terms of classification performances. However, the proposed framework can be more effective in terms of computational costs.

Chapter 4

Hardware Implementation

A considerable amount of applications relies on embedded systems that are expected to yield online predictions [41, 95–100]; i.e., the prediction function should process in real-time each new sample that feeds the system. In this regard, resource-constrained, low-power embedded systems provide a challenging scenario for the deployment of such predictors. Single layer feed forward neural networks such as the models introduced in Chapters 2 and 3 represent an effective option when targeting implementations on low-cost devices, because, in general, they achieve a satisfactory trade-off between prediction accuracy and computational complexity. The prediction function eventually consists in a weighted sum of nonlinear activation functions, where the number of terms equals the number of neurons.

Actually, deep learning paradigms may offer improved performance in terms of prediction accuracy; on the other hand, the hardware implementation of the corresponding decision function still does not fit resource-limited devices [44, 101, 102].

As introduced in Chapter 2, from a hardware perspective, the most interesting configuration for SLFNs is possibly the one that exploits only hard-limiter (threshold) activation functions in the hidden layer. This setup sharply reduces the overall circuit complexity in the predictor digital implementation. Nonetheless, one should face two crucial drawbacks. First, the (offline) learning phase cannot be supported by conventional algorithms, as the hard-limit function is non-differentiable [103]. Second, the threshold activation function in general affects the generalization ability of the eventual SLFN, which might require a large number of neurons to achieve a satisfactory accuracy.

As pointed out in subsection 2.5.2 the theoretical framework of Extreme Learning Machines [6, 32] with a suitable management of the sampling procedure offer a viable solution to these problems. The resulting learning procedure achieved two goals. First,

to provide a suitable tool to train a SLFN based on hard-limiter activation function. Second, to provide a training algorithm that can effectively balance generalization performance and the number of neurons in the hidden layer. Actually, the latter aspect represents a major achievement when targeting resource-constrained devices.

This chapter exploits the outcomes of the research presented in subsection 2.5.2 to focus on the design of the digital architecture that can efficiently support the implementation of the trained SLFN. The emphasis is on low-cost, low-performance devices such as CPLDs and low-end FPGAs. Accordingly, the present work analyzes two different design strategies that can fit such scenario. The first strategy primarily addresses an efficient utilization of the hardware resources, for this reason, all the design choices aim to minimize the allocation of resources. The second strategy, conversely, reduces the latency through the parallelization of the most time consuming part in the neural network, i.e. the computation of the scalar product $\boldsymbol{x} \cdot \hat{\boldsymbol{r}}$ in equation (2.18). As a result, both strategies implement design criteria that remove the need of any multiplier in the eventual architecture.

In principle, when the hidden layer is not subject to any selection or pruning criteria, all the ROM memories employed to store random parameters $\{\boldsymbol{r}, \chi\}$ can be replaced by pseudo random number generators (PRNG). This design choice dramatically reduces the requirements of area for the deployment of the eventual predictors. Unfortunately, almost all the state of the art strategies suitable to improve the trade-off between computational complexity and generalization performances summarized in subsection 2.5.4 are discarded by the constraints imposed by the use of PRNG. However, it is possible to adapt the architecture to handle a subset of selection strategies with a minimal computational overload.

The proposed research differs in the goals from the work described in [62, 63, 99], where VLSI implementation was the main target, and [100], where the overall architecture was designed for online training. Patil et al. [62] indeed relied on a tristate activation function and a pruning procedure to reduce the circuit complexity of the eventual ELM predictor. However, the overall model targeted analog implementations. Binary weights have been also applied to convolutional neural networks to limit computational complexity [44]; the resulting networks, though, bring about approximations that are unsuitable for SLFNs, as the latter ones involve a small number of parameters.

Recently, different authors addressed the problem of an efficient storing of the random hidden layer weights. Taking advantage of the hidden layer weights's random nature, both [62, 104] proposed a rotation scheme that enables the storage of a small number of hidden parameters. These approaches differ from the proposed one because they target a

mixed analogical-digital design and the advantages of these methods become less evident in a fully digital implementation, that is the target of this thesis.

4.1 Digital Implementation

This chapter targets the design of a modular digital architecture that can conveniently implement the decision function (2.11) for the specific case of neurons' activation (2.18):

$$f(\boldsymbol{x}) = \sum_{i}^{N} \beta_n \cdot sign(\boldsymbol{x}^t \hat{\boldsymbol{r}}_n - r_n) = \sum_{i}^{N} \beta_n \cdot sign(I_n) \tag{4.1}$$

As a major consequence of targeting resource-constrained devices, such as CPLDs and low-end FPGA, the overall design strategy requires one to tackle a few implementation issues to the purpose of achieving a proper balance between accuracy, resource allocation and computational complexity. In the following, it is assumed that both the i-th feature x_i and the n-th weight β_n are represented as a B-bit fixed point, 2's complement number.

The first critical issue in (4.1) concerns the computation of the input I to the activation function:

$$I(\boldsymbol{x}, \hat{\boldsymbol{r}}_n, r_n) = \|\boldsymbol{r}_n\|(\boldsymbol{x}^t \hat{\boldsymbol{r}}_n - r_n) \tag{4.2}$$

A remarkable properties of combining I with threshold units is that the quantity $\|\boldsymbol{r}\|$ that characterizes eq. (2.18) does not affect the sign of the activation I reducing automatically the number of computations. Operation (4.2) involves D multiplications and D sums; moreover, it should be repeated N times (one for each neuron). Actually, tough, one can avoid multiplications by imposing the following constraint on the random weights $\hat{r}_{d,n}$:

$$\hat{r}_{d,n} = \texttt{sign}(r_1) 2^{-\lceil r_2 \rceil} = s \cdot 2^{-k} \tag{4.3}$$

where $r_1 \in \{-1, 1\}$ and $r_2 \in N^+$ are random quantities. It worth to note that this constraint does not violate any of the constraints in ELM theories [33]. As a result, the computation of the single term $x_d \cdot \hat{r}_{dn}$ can be completed with a shift operation ($x_d >> k_{dn}$), which should be followed by a sign conversion when $s_{dn} < 0$ (i.e., $\hat{r}_{dn} < 0$).

In principle, to apply sign conversion to the shifted term one needs a bitwise *not* operator followed by a sum with a constant term, whose value is '1 Least Significant Bit' (LSB).

However, when implementing (4.1) one knows in advance how many weights r_{dn} are negative. In this regard, let W be the number of negative r_{dn}. Then -when computing $\boldsymbol{x}\hat{\boldsymbol{r}}_n$- the single W terms '1 LSB' can be replaced by a single cumulative term ζ. As a result, one has

$$\boldsymbol{x}^t\hat{\boldsymbol{r}}_n - r_n = \sum_d \bar{x}_d - r_n + \zeta_n \tag{4.4}$$

where

$$\bar{x}_d \begin{cases} x_d >> k_{dn}, & \text{if } s_{dn} = 1 \\ \texttt{NOT}(x_d >> k_{dn}), & \text{if } s_{dn} = -1 \end{cases}$$

Such approach eventually simplifies the design of the corresponding digital device.

The second critical issue in general concerns the computation of $\phi(I)$. Again, this operation must be completed N times (one for each neuron). A standard choice for the activation function ϕ is the sigmoid. In literature, the hardware implementation of such function is achieved either by exploiting piecewise linear approximations or by relying on lookup tables [105]. The former solution is usually preferred, as lookup tables require large memories. Piecewise linear approximation, though, always involves a multiplication and a sum. Instead, threshold activation function in a digital implementation consists in the fetch of the most significant bit.

Moreover, a SLFN based on the hard-limiter activation function is characterized by a further advantage. The term $\beta_n \cdot \phi(I)$ in (4.1) can be computed without relying on a multiplier, since $\phi(I)$ only sets the sign of β_n.

The architecture supporting the digital implementation of the trained SLFN can be organized according to three main modules:

- The first module, *Input*, stores the components (features) of the pattern $\boldsymbol{x}$ to be classified.
- The second module, *Neuron*, computes -for the n-th neuron- the quantity $a_n = sign(\boldsymbol{x}^t\hat{\boldsymbol{r}}_n - r_n)$.
- The third module, *Output*, supports the output layer and incrementally finalizes the computation of (4.1). It activates as soon as *Neuron* provides a_n for the n-th neuron.

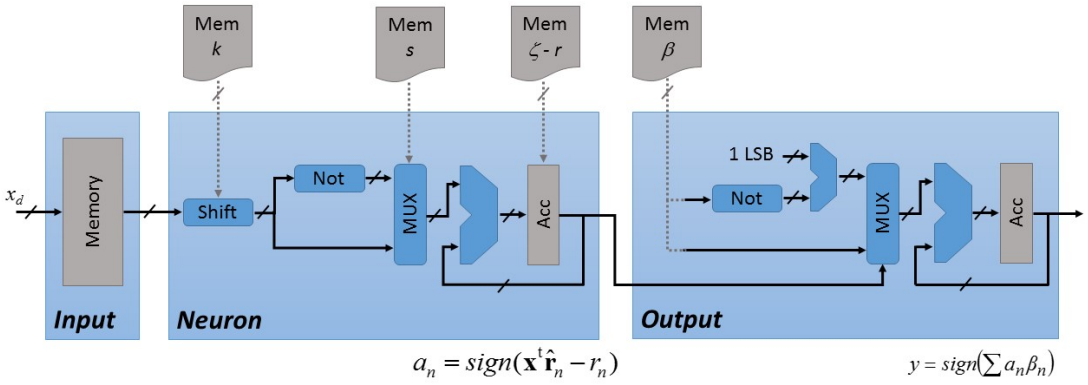


FIGURE 4.1: Architecture for the digital implementation of the predictor: sequential approach architecture;

The architecture should indeed include a finite-state machine ruling the whole control process and storage resources to host the following quantities generated by the learning procedure: $\{\{(s,k)_d; d = 1,..,D\}_n, \zeta_n, l_n, \beta_n; n = 1,..,N\}$.

Such overall organization can result in two different implementation approaches. The first approach relies on a *Neuron* module that sequentially processes the features x_d, as in [9]. In the second, alternative approach, the *Neuron* module processes in parallel all the D components of the input $\boldsymbol{x}$.

4.1.1 Serial Neuron Implementation

The architecture implementing the sequential approach is outlined in Fig. 4.1. The internal design of the three main modules is organized as follows:

Input: a memory that hosts the D feature values. The module supports a serial input and provides as output a single feature x_d per read-cycle.

Neuron: this module computes the quantity a_n step-by-step. First, when a feature x_d feeds the module, a shift operation works out the term $x_d >> k_{dn}$, as per (4.3); the shift circuit is fed by a memory that stores the integers k_{dn}. Second, the corresponding sign bit s_{dn} drives a multiplexer, which selects the output of the bitwise not operator only when $s = -1$, i.e., when $x_d \cdot \hat{r}_{dn}$ involves a sign conversion. The final add-and-accumulate block is entitled to carry out incrementally the computation of the completed term (4.4). This goal is obtained by initializing the accumulator to the value $(\zeta_n - r_n)$ before the

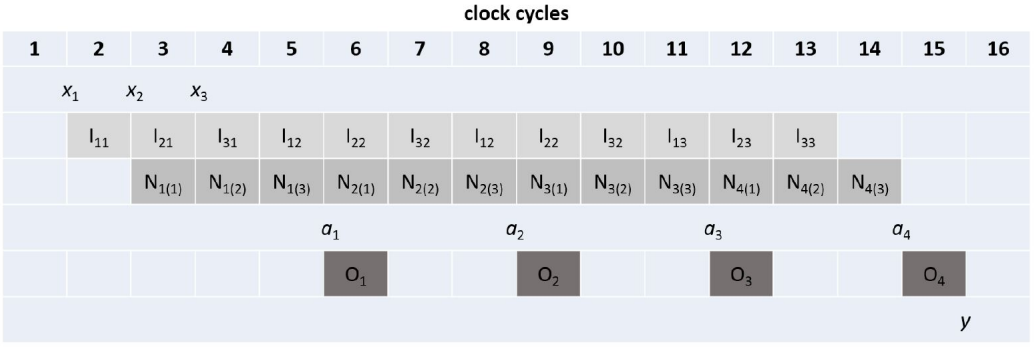


FIGURE 4.2: Example of processing flow for the proposed sequential architecture;

processing of x_1 starts; a memory supports such procedure. The module provides as output the signal that marks the sign bit in the accumulator, i.e., a_n.

Output: this module is designed to obtain the prediction $y(\boldsymbol{x})$ step-by-step. As $a_n \in \{-1, 1\}$, the quantities $a_n \cdot \beta_n$ are computed by exploiting a sign unit, which supports the conversion of β_n, if needed. The weights β_n are stored in the corresponding memory. An add-and-accumulate block supports the incremental computation of y. The module gives as output the signal that corresponds to the sign bit in the accumulator, i.e., the prediction.

Figure 4.2 shows an example of the processing flow for $D = 3$ and $N = 4$ (i.e., four neurons in the hidden layer). The scheme outlines the functional activity of the following single-cycle processes:

- I_{dn}: input cycle. The *Input* module provides as output the d-th feature; the storage units provide to the datapath the parameterization set by $\hat{r}_{dn}$.
- $N_{n(d)}$: processing cycle of the *Neuron* module. The module instances the n-th neuron and processes the d-th feature, thus computing $x_d \cdot \hat{r}_{dn}$.
- O_n: processing cycle of the *Output* module. The module processes a_n, thus computing $a_n \cdot \beta_n$.

In the proposed example, features x_1, x_2, and x_3 are stored in the memory hosted by the *Input* module at the rising edge of clock cycles 2, 3, and 4, respectively. Accordingly, I_{11} can be completed within clock cycle 2 and the *Neuron* module can start the incremental computation of a_1 at clock cycle 3 ($N_{1(1)}$). At the rising edge of clock cycle 6 the accumulator hosted by the *Input* model stores a_1, as $N_{1(D)}$ has been completed. Thus, within clock cycle 6 the *Output* module computes $a_1 \cdot \beta_1 (O_1)$. Eventually, O_N is executed at clock cycle 15 and the eventual prediction y is available in the accumulator hosted by

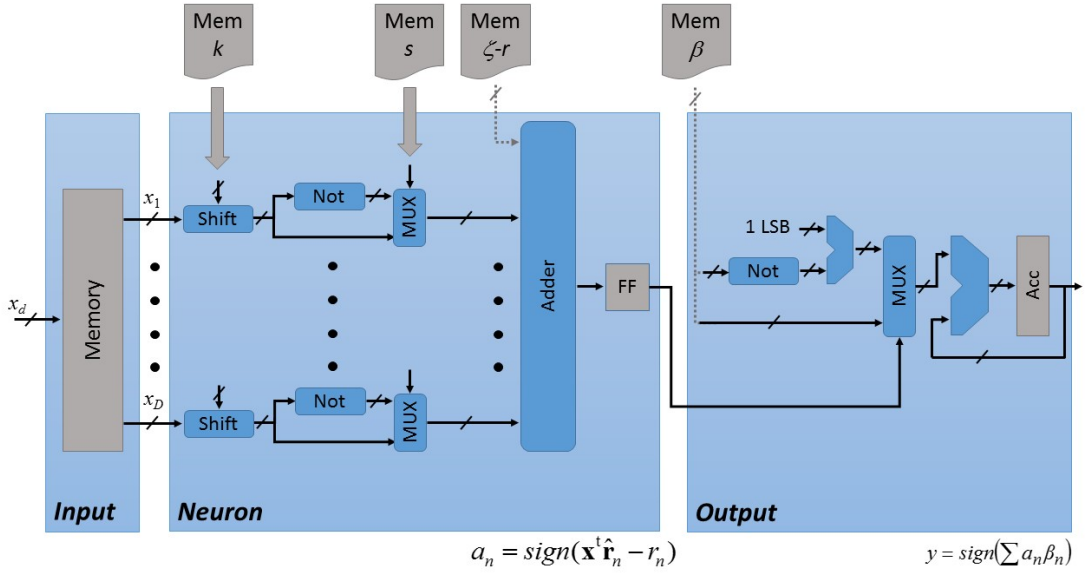


FIGURE 4.3: Architecture for the digital implementation of the predictor: parallel approach architecture;

the *Output* module at clock cycle 16. Overall, this approach ensures a prediction results in $(D \cdot N + 3)$ clock cycles, starting from the fetch of the first feature.

4.1.2 Parallel Neuron Implementation

Figure 4.3 outlines the architecture that exploits the fully parallel *Neuron* module.

The internal design of the main modules changes as follows:

1. ***Input***: the module still contains a memory with serial input that hosts D features. However, it now provides as output all the D features in parallel per read-cycle.
2. ***Neuron***: this module computes the quantity a_n in a single clock-cycle. Accordingly, the features x_d are processed in parallel by D instances of the block 'shift-not multiplexer' already described above. An adder with $D + 1$ inputs finalizes the computation of a_n; the additional input refers to the quantity $(\zeta_n - r_n)$. A flip-flop (FF) is set to store the signal that marks the sign bit in a_n.
3. ***Output***: this module is still designed to obtain the prediction $y(\boldsymbol{x})$ step-by-step.

Such design supports a fully pipelined processing, as showed in Fig. 4.4, which -as above- refers to an example of the processing flow for $D = 3$ and $N = 4$. The pipeline operation involves the following three single-cycle processes:

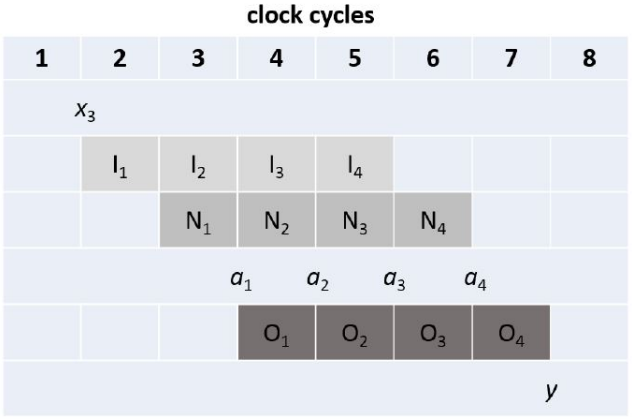


FIGURE 4.4: Example of processing flow for the proposed parallel architecture;

- I_n: input cycle. The *Input* module provides as output the D features in parallel; the storage units provide to the datapath the parameterizations set by $\{\hat{r}_{n1}, ..., \hat{r}_{nD}\}$.
- N_n: processing cycle of the *Neuron* module. The module instances the n-th neuron, thus computing a_n.
- O_n: processing cycle of the *Output* module. The module processes a_n, thus computing $a_n \cdot \beta_n$.

In this case, the processing can start only when the D-th feature has been stored in the memory hosted by the *Input* module. Accordingly, in the proposed example the processing flow is triggered by the availability of x_3 at the input port (rising edge of clock cycle 2). At the rising edge of clock cycle 4 the *Neuron* module stores a_1 in the corresponding FF; indeed, a_N is ready after $(N-1)$ clock cycles. The eventual prediction y is available in the accumulator hosted by the *Output* module at clock cycle 8. Overall, this approach ensures a prediction results in $(N+3)$ clock cycles, starting from the fetch of x_D.

4.1.3 Comparing the Two Designs: Area Utilization vs Latency

The two different approaches to the design of the trained predictor can be compared according to the following attributes:

- area utilization: this parameter actually mostly depends on the design of the *Neuron* module. In the fully pipelined approach (Fig. 4.3), the dimensionality D of the input space has a direct impact on the resources occupied by the instance

of the n-th neuron. The corresponding hardware implementation should support parallel processing of the D features; in addition, the *Adder* block should process $D + 1$ inputs. Conversely, in the sequential approach of Fig. 4.1 the resource utilization of the *Neuron* module is not affected by D. Clearly, also the number of bits B plays a role in determining the area covered by the *Neuron* module. In fact, B is expected to have more impact in the sequential design. Nonetheless, D, B and the number of neurons N influence -in both the approaches- the size of the memories included in the architecture. In this regard, one should consider that the size of memory hosted in the *Input* module grows as $D \times B$; besides, the memories entitled to store parameters s and k both grow as $D \times N$.

- latency: to the purpose of assessing this parameter, let T_s and T_p, respectively, be the clock periods of the hardware implementations resulting from the sequential architecture (Fig. 4.1) and the fully pipelined architecture (Fig. 4.3). As a result, the latency associated to the prediction when adopting the first approach is $(D \cdot N + 3) \cdot T_s$, while the pipelined approach yields a prediction in $(N + 3) \cdot T_p$. This in turn means that -given N- one would need $T_s \simeq T_p/D$ to obtain a predictor based on the sequential architecture that has the same latency of a predictor based on the fully pipelined architecture. On the other hand, one should consider that in the sequential approach the processing can start when the feature x_1 has been stored. In the fully pipelined approach, in contrast, processing can start only when the last feature, x_D, has been stored. Such aspect might become relevant when $D >> N$.

Overall, the sequential approach best fits scenarios in which effective area utilization has the priority over latency. Thus, such approach is expected to be effective when one wants to maximize the quantity $D \times N$, given N. Conversely, the fully pipelined approach is expected to support implementations that suitably balance $D \times N$ and latency.

4.2 Pseudo-random Number Generator Based Architecture

Storing of the hidden neurons parameters is one of the main factors of area consumption when SLFNNs' deployment is considered. In principle, it is necessary to store $D+1$ real numbers for each neuron.

As a major consequence of constraint (4.3), the hidden parameters $\hat{r}_{dn}$ are described by the couples $\{k, s\}_{dn}$ (eq. 4.3) where s is a bit and the value k is an integer number that should be always smaller than B i.e. the maximum number of shift operation in a

register of size B. In practice, each couple $\{k, s\}$ needs only $1 + log_2(B)$ bits. However, the size of these two memories grows as $N \times D$ and it rapidly becomes one of the main sources of area consumption.

Interestingly, the only requirements about the pairs $\{k, s\}$ is that they are randomly generated; therefore, it is possible to generate these numbers using a pseudo random number generator (PRNG) that, given a fixed initial condition, produces always the same chaotic sequence of numbers. In this way, the requirement of storing $N \times D$ parameters becomes the requirement of implementing an efficient sequence generator of $N \times D$ values.

Linear feedback shift register (LFSR) [106] is a simple and compact realization of such models. Given a shift register of size m and a *xor* based feedback function that receives in input few bits of the register, it is possible to obtain a sequence of non-repeating numbers with period $2^m - 1$. For each size m, standard tables [107] provide the bit index that should be used in the *xor* feedback function.

In general, the entire set $\{\boldsymbol{K}, \boldsymbol{s}, r\}$ can be set randomly; however, this chapter refers to the training algorithm proposed in subsection 2.5.2 because it provides a better trade-off between generalization performances and hidden layer size. For this reason, parameter r is subject to a selection criteria and need to be stored in a ROM memory; the extension to the general case is straightforward.

Two design constraints drive the size m of the shift register: 1) m should be big enough to guarantee that, at each clock cycle, enough bits are provided to represent the hidden parameters ; 2) m should be such that, the period of the random sequence is greater than $N \times D$, i.e. the amount of hidden parameters.

According to the selected architecture the ratio between the constraints changes. In serial neurons approach (Fig. 4.1) size of the random generator is subject to the following constraints: at each clock cycle a couple $\{k, s\}$ should be generated then $m \geq 1+log_2(B)$ bit are necessary, at the same time m should be such that $2^m - 1 \geq N \times D$ to avoid identical configurations. The fully pipelined approach (Fig. 4.3), instead, requires D couples $\{k, s\}$ at each clock cycle, then the first condition becomes $m \geq D + D * log_2(B)$, while the constraint about sequence period becomes less stringent $2^m - 1 \geq N$. Should be noted that the size of the register is, in both the cases, negligible respect to the number of elements that should be stored; in fact, the first constraints correspond to the output of the ROM in a single clock cycle, while the second one is satisfied by an exponential function of m.

In principle, the use of LFSR to replace ROM memories prevent the use of pruning or selection techniques because the random sequence cannot be controlled. On the other

hand, shrinking techniques are particularly useful in cases where it is necessary to avoid large dimensions of the hidden layer size.

Fortunately, given a register state, the LFSR produces always the same sequence of numbers. In other words, the seed of LFSR corresponds to its output. This characteristic enables the use of a strategy for serial architecture (Fig. 4.1) where only the first pair $\{s_{1d}, k_{1d}\}$ is stored, while the other parameters can be generated using the LFSR. Considering serial architectures the overload is negligible in term of area consumption; in fact, for each neuron it is possible to store only the starting point of the PRNG, i.e. $\{s_{1d}, k_{1d}\}$, reducing the number of parameters to be stored from $N \times D$ to N. Unfortunately, the same reasoning does not hold for the fully pipelined case where the starting point of each sequence would correspond to the sequence itself because each neuron needs to be computed in a single clock cycle.

4.3 Experimental Validation

The experimental section provides a detailed comparison about the implementation of the proposed models into low cost devices such as CPLD and low end FPGAs. The experimental tests are divided into three main parts. In the first one, tests regarding the implementation of serial architecture are provided, with particular attention on the link with other state-of-the-art proposal. Following a detailed comparison between the architecture proposed in subections 4.1.1 and 4.1.2 is presented. Finally, the results obtained replacing traditional ROM memories with pseudo random generator are depicted in section 4.2.

4.3.1 Serial Architecture Deployment

The architecture given in Fig. 4.1 was tested on two low-resources devices, namely, Altera CPLD 5M1270Z, and Altera CPLD 5M2210Z[1]. The two platforms differ in the number of available macrocells: 5M2210Z almost offers a double number of macrocells as compared with 5M1270Z. Analysis and synthesis of the proposed architectures were accomplished using Quartus Prime software.

The first test aimed at evaluating the ability of the proposed architecture to fit such CPLDs. In practice, the area occupancy of the architecture is determined by D (input dimensionality) and N (size of the hidden layer), as they set the capacity of the involved memories. Thus, Fig. 4.5 gives D on the x axis; the y axis provides given D the size N that would correspond to the full area occupancy on the device when deploying the

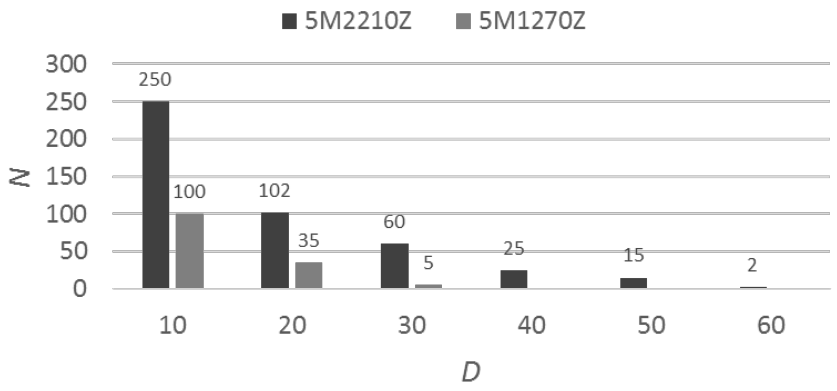


FIGURE 4.5: Configurations that would correspond to full area occupancy when deploying a predictor based on the proposed architecture.

predictor. In this figure the black-filled bars refer to the 5M2210Z, while the gray-filled bars refer to the 5M1270Z. All the implementations adopted a 12 bit fixed-point representation. In terms of power consumption, such implementations needed, respectively, about 75 mW on the CPLD 5M2210Z and about 48 mW on the CPLD 5M1270Z (with a $33MHz$ clock frequency in both cases). These outcomes confirm that low-resources devices can support predictors trained on low-dimensionality problems. Indeed, one can exploit FPGAs when targeting problems characterized by medium or large dimensionality.

The second test aimed at providing a direct comparison with the state-of-the-art design for the digital implementation of ELM classifiers [41]. In [41], the predictor employed a standard ELM based on sigmoid function; training was supported by a customized procedure that enabled pruning mechanisms. The corresponding digital implementation stemmed from a design strategy that combined parallel and pipelined processing, ensuring a prediction results in $D \times N$ clock cycles.

To provide a fair comparison, the test involved the implementations discussed in [41], which were deployed on CPLD 5M1270Z. The predictor implementations covered three benchmarks, which in turn spanned as many settings of D. For each benchmark, two predictors were deployed, which differed in the setup of N: one predictor stemmed from a conventional training strategy, while the other one used the training strategy with a pruning mechanism, thus shrinking the size of the hidden layer.

Table 4.1 reports on the outcomes of this test, and compares, given D and N, the area utilizations required by the proposed HE and the architecture presented in [41], respectively. The second and the third columns of the table refer to the predictors trained on Breast Cancer Wisconsin ($D = 9$). The fourth and the fifth columns refers

[1]https://www.altera.com/products/cpld/max-series/max-v/overview.html

TABLE 4.1: Area Occupancy - CPLD 5M1270Z

D	9		34		81	
N	4	10	22	80	18	70
HE	17%	22%	50%	$> 100\%$	73%	$> 100\%$
[41]	40%	40%	86%	$> 100\%$	95%	$> 100\%$

to the predictors trained on Ionosphere ($D = 34$). The last two columns refer to the predictors trained on MNIST ($D = 81$). The reported occupancy actually gives the area covered without the *Input* module (as per Fig. 4.1), as the original classifier designed in [41] did not include such module and adopted serialization to fetch input patterns.

Numerical outcomes confirm the reliability of the proposed digital design. The HE predictor always occupied less area than the corresponding predictor based on the architecture designed in [41]. In a few cases, though, both the architecture deployed an implementation that did not fit the area provided by the CPLD. In this regard, it is interesting to note that HE would reach full area occupancy with $N = 55$ when $D = 34$, and with $N = 25$ when $D = 81$.

4.3.2 Serial/Parallel Implementations Comparison

The architecture shown in Fig. 4.1 and Fig. 4.3 were tested on two low-resources devices, namely, the Altera CPLD 5M2210Z and the Altera FPGA EP4CGX15BF14A7[2], which does not embed multipliers. The low end FPGA was selected with the aim of relaxing the constraint about limited resources. The tests aimed at evaluating the ability of the proposed architectures to fit low-resources devices, when the constraint about area consumption are relaxed with respect to the previous experimental setup (subsec. 4.3.1).

Figure 4.6 presents the outcomes of the tests based on the CPLD. Figure 4.6(a) refers to the hardware implementation of the sequential approach (Fig. 4.1). The figure gives N on the x axis, while the y axis marks the dimensionality of the input space, D, that would correspond to the full area occupancy on the device when deploying the predictor. The test involved four different configurations of $N = \{25, 50, 75, 100\}$. The figure indeed plots the outcomes obtained with three different settings for the number of bits B: the black markers refer to the tests based on $B = 8$; the light gray markers refer to the tests based on $B = 10$, the dark gray markers refer to the tests based on $B = 12$. All the implementations required about $130mW$ in terms of power consumption, with clock frequency set to $55MHz$. The tests showed that the sequential approach was able to support a predictor designed to deal with a 90-dimensional problem, under the configuration $\{N = 25, B = 8\}$. In fact, D reduces to 60 when one needs a 12-bit representation of data to save accuracy. Indeed, it is worth to note that a 25- dimensional

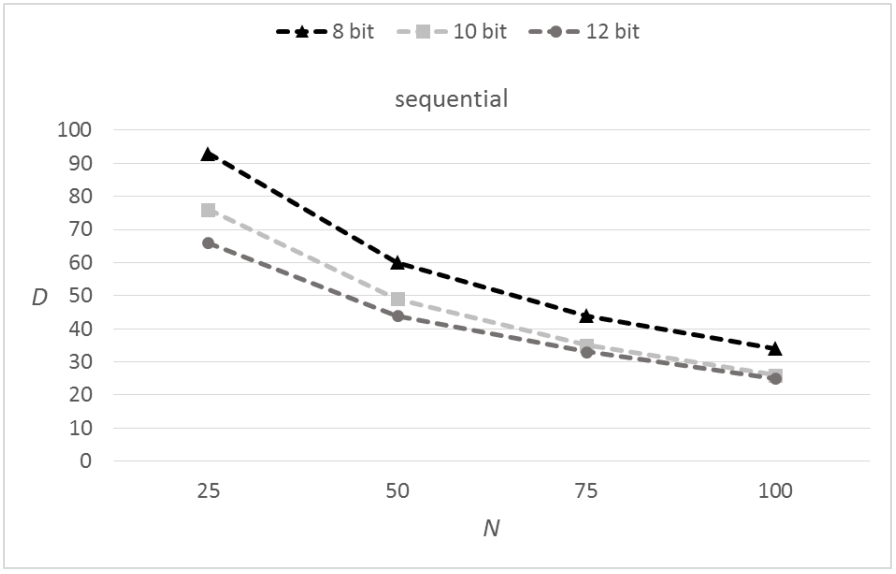
TABLE 4.2: Implementation on CPLD: D_{25}/D_N as a function on N

	sequential		fully pipelined	
N	B=8	B=12	B=8	B=12
50	1.55	1.50	1.31	1.22
75	2.11	2.00	1.52	1.46
100	2.73	2.64	1.80	1.69

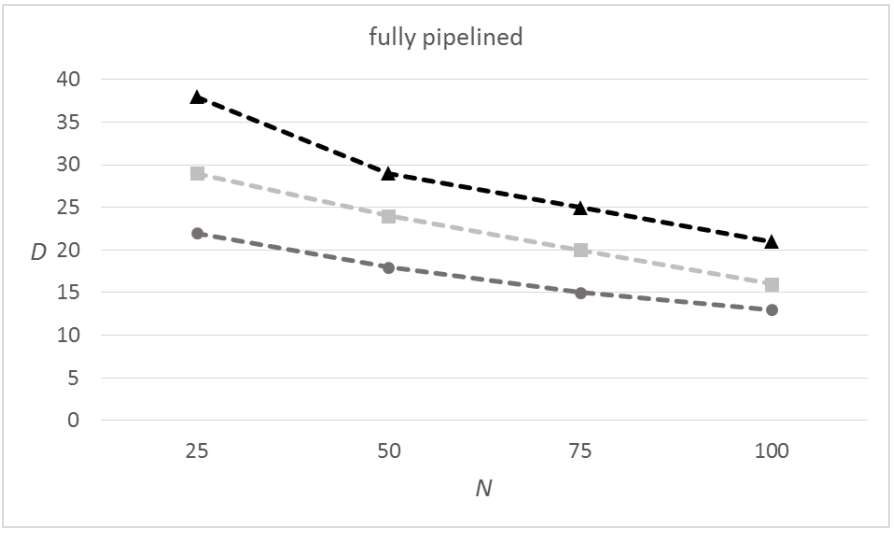
problem could be managed under the configuration $\{N = 100, B = 12\}$. This is a remarkable result for an implementation on a low-end, low-cost device such as the CPLD 5M2210Z. Figure 4.6(b) refers to the hardware implementation of the fully pipelined approach (Fig. 4.3); the plot adopts the same format of Fig. 4.6(a). As expected given N the fully pipelined approach imposes stricter constraint on D with respect to the sequential approach. Nonetheless, the figure shows that one would be able to implement a predictor designed to deal with a 37-dimensional problem under the configuration $\{N = 25, B = 8\}$; indeed, a 20-dimensional problem could still be managed under the configuration $\{N = 100, B = 8\}$. Moreover, the fully pipelined approach proved able to achieve interesting performance in terms of latency, as the clock frequency was set to $35MHz$ in all the implementations. This in turn means that $T_s = T_p/1.5$, independently of D, while one would need $T_s = T_p/D$ to obtain an implementation of the sequential approach that can compete with the corresponding implementation of the fully pipelined approach. Obviously, one should also take into account the trade-off between latency and power consumption, which in the fully pipelined approach is affected by D. In the tested implementations, power consumption ranged from a minimum of $150mW$ (with $D = 13$) to a maximum of $250mW$ (with $D = 38$).

Overall, it is interesting to assess how the quantity $D \times N$ changed as N grew, for both the approaches. To this purpose, let D_N denote the value took by D when the number of neurons in the predictor was set to N. Accordingly, Table 4.2 provides the value of the quantity D_{25}/D_N for $N = \{50, 75, 100\}$. The table is organized as follows: the first columns gives N; the second and the third column refer to the sequential approach and provide the value of D_{25}/D_N under the configuration $B = 8$ and $B = 12$, respectively; the same format is used the fourth and fifth column, which refer to the fully pipelined approach. Numerical results show that in the fully pipelined approach D dropped slower than in the sequential approach, as N grew. Thus, while the sequential approach -given N- can maximize the quantity $D \times N$, the fully pipelined approach scales better as N grows.

Figure 4.7 presents the outcomes of the tests based on the FPGA. As above, Fig. 4.7(a) refers to the hardware implementation of the sequential approach, while Fig. 4.7(b) refers to the hardware implementation of the fully pipelined approach. Both the plots adopt the



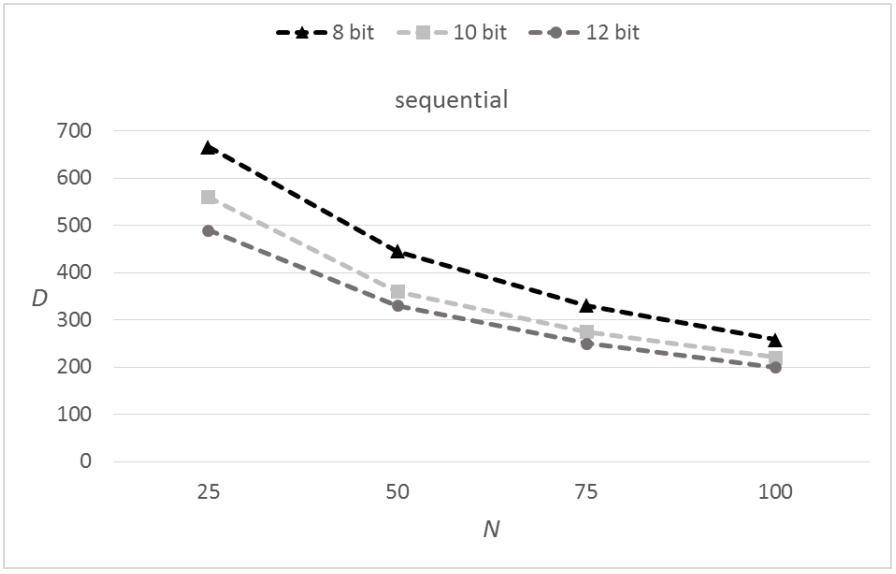
(a)



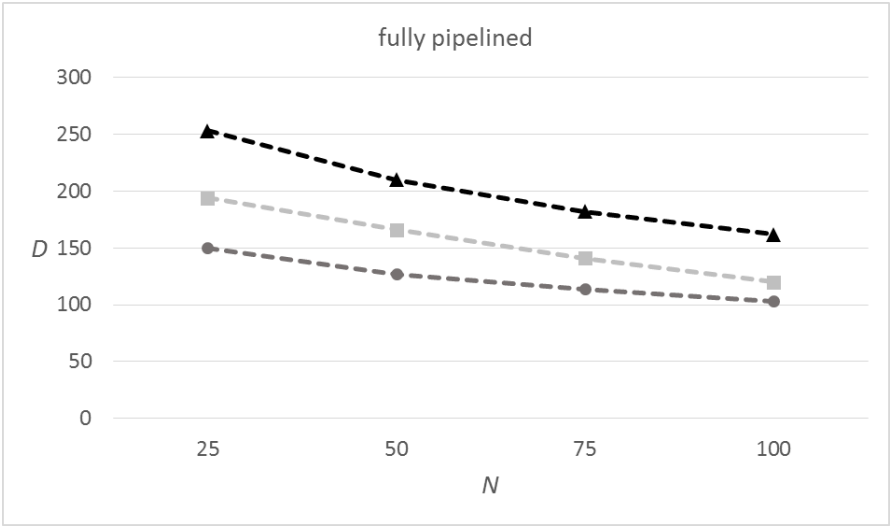
(b)

FIGURE 4.6: Tests on CPLD: configurations that would correspond to full area occupancy when deploying the predictor: (a) sequential approach; (b) fullpipelined approach.

same format of Fig. 4.6(a). Clearly, implementations on FPGA allow both approaches to support predictors that can deal with $D > 100$; for example, Fig. 4.7(a) shows that by adopting the sequential approach one could tackle 500-dimensional problems. Actually, it is interesting to compare the implementations on FPGA with the implementations on CPLD in terms of power consumption and latency. The predictors supported by the sequential approach recorded a power consumption of about $130mW$ on the FPGA; thus, this parameter did not change with respect to CPLD implementations. Nonetheless, the clock frequency on FPGA raised up to $75Mhz$. Implementations based on the fully pipelined approach required a power consumption of $120mW$ on the FPGA, with the



(a)



(b)

FIGURE 4.7: Tests on FPGA: configurations that would correspond to full area occupancy when deploying the predictor: (a) sequential approach; (b) fullpipelined approach.

clock frequency set to $50Mhz$. Thus, FPGA implementations confirmed that $T_s \simeq T_p/1.5$.

As above, Table 4.3 provides the value of the quantity D_{25}/D_N for $N = \{50, 75, 100\}$ in the case of the FPGA implementations. This table adopts the same format of Table 4.2. Numerical outcomes confirmed that also on FPGAs the fully pipelined approach scales better as N grows.

[2]Altera FPGAs https://www.altera.com/products/general/fpga.html

TABLE 4.3: Implementation on FPGA: D_{25}/D_N as a function on N

	sequential		fully pipelined	
N	B=8	B=12	B=8	B=12
50	1.49	1.48	1.20	1.18
75	2.01	1.96	1.39	1.31
100	2.58	2.45	1.56	1.45

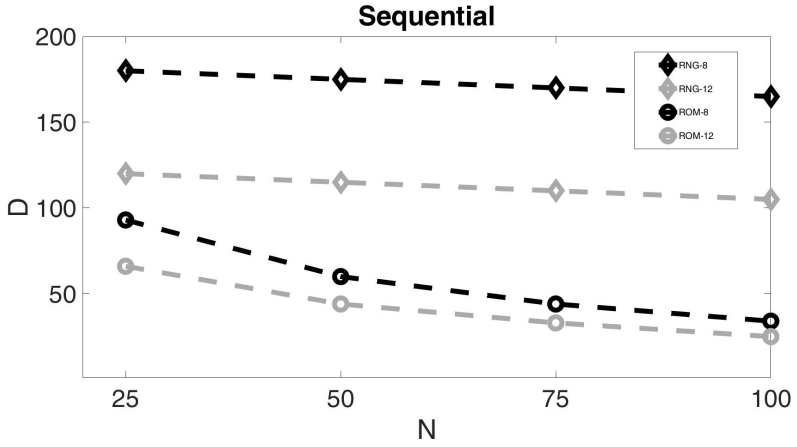
4.3.3 ROM/PRNG Implementations Comparison

The purpose of this experimental campaign is the quantification of the gain introduced by the use of PRNG to replace ROM memories, as per section 4.2. The architectures shown in Fig. 4.3 and 4.1 were tested on two low-resources devices, namely, the Altera CPLD 5M2210Z and the Altera FPGA EP4CGX15BF14A7 which does not embed multipliers, using the same setup of section 4.3.2. The tests aimed at evaluating the ability of the proposed architectures to fit low-resources devices.

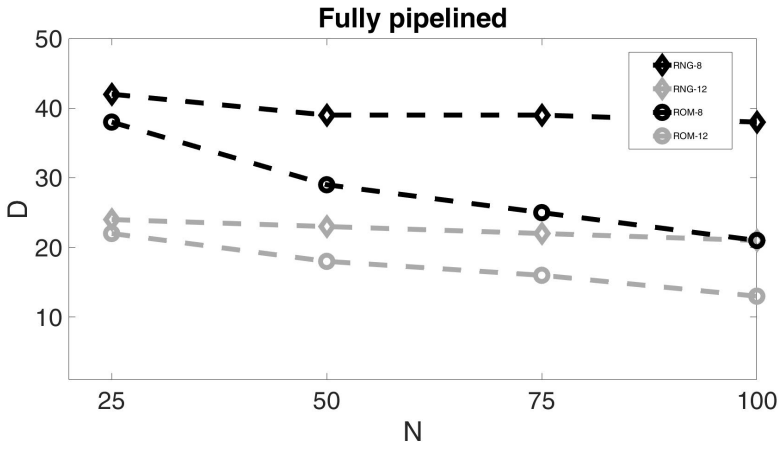
Figure 4.8 presents the outcomes of the tests based on the CPLD. Figure 4.8(a) refers to the hardware implementation of the sequential approach. The figure gives N on the x-axis, while the y-axis marks the dimensionality of the input space, D, that would correspond to the full area occupancy on the device when deploying the predictor. The test involved four different configurations of $N : \{25, 50, 75, 100\}$. The figure indeed plots the outcomes obtained with two different settings for the number of bits B: the black lines refer to the tests based on B = 8; the light grey lines refer to the tests based on $B = 12$. Circle markers refer to standard ROM implementation, while diamonds stand for PRNG. The clock frequency was $20MHz$ for all the implementations. The tests showed that configurations based on PRNG outperform always the ones based on standard memory. Furthermore, with the growth of N, the gap increase substantially. Interestingly, configuration based on 8 bit arithmetics can support the implementation of a network with $N = 100, D = 165$.

Figure 4.8(b) refers to the hardware implementation of the fully pipelined approach; the plot adopts the same format of Fig. 4.8(a). As expected -given N- the fully pipelined approach imposes stricter constraints on D with respect to the sequential approach. As in the previous case, PRNG based solutions, always outperform the corresponding counterparts in term of area occupancy. This advantage becomes particularly evident when $N = 100$, in fact, for ROM based solution $D = 21$ is the limit, while PRNG can manage classifiers with D up to 38.

Figure 4.9 inherit the format from figure 4.8. The result refers to the implementation of the architectures on FPGA where the clock frequency was $50MHz$ and the setting for D is $\{100; 500; 1000\}$. Outcomes confirm the trend shown in the previous simulations. The



(a)



(b)

FIGURE 4.8: Tests on CPLD: configurations that would correspond to full area occupancy when deploying the predictor with standard rom memories (diamond markers) or PRNG (circle markers): (a) sequential approach; (b) fullpipelined approach.

most important observation is that implementation with PRNG allows configuration that can be adopted for almost any application; in fact, a predictor based on the serial neuron implementation with a size of the hidden layer $N = 1000$ can handle input data with dimensions bigger than 1200. Instead with fully pipelined architecture input dimensionality D is limited to 200, but the latency becomes $\simeq 200$ times smaller.

4.4 Concluding Remarks

This chapter presented digital architectures designed to implement random basis neural networks based classifier on low-cost devices. The research focused on implementations that adopt the hard-limiter activation as activation function. This configuration can take advantage of two attributes: 1) it can lead to hardware-friendly design, and 2)

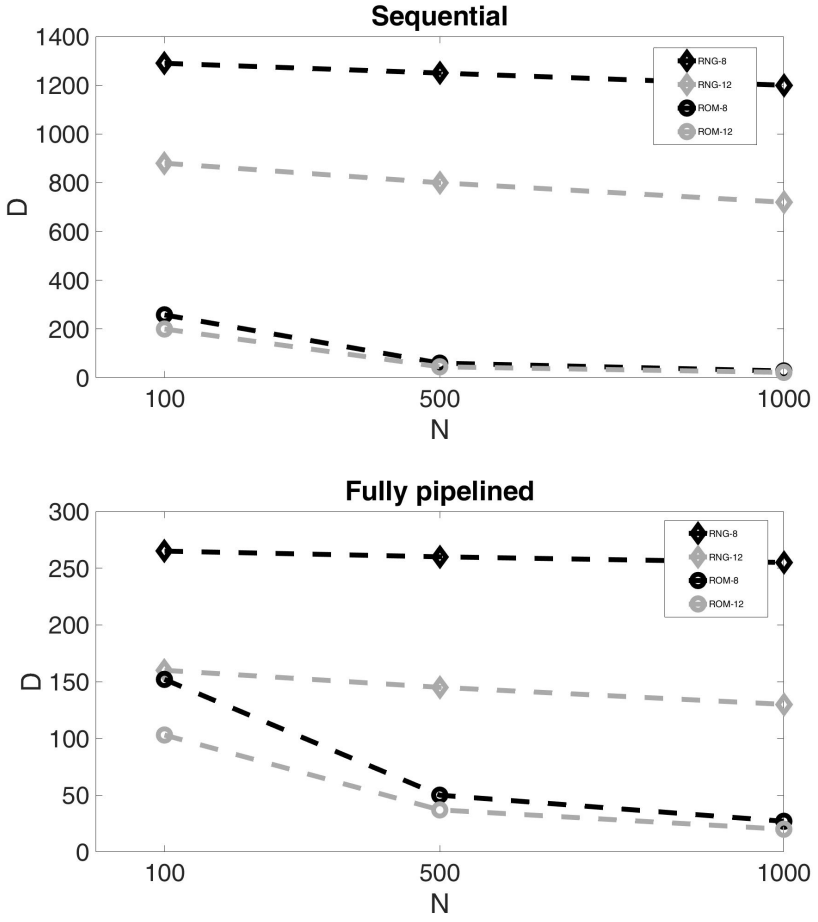


FIGURE 4.9: Tests on FPGA: configurations that would correspond to full area occupancy when deploying the predictor with standard rom memories (diamond markers) or PRNG (circle markers): (a) sequential approach; (b) full-pipelined approach.

the literature provides learning schemes that can suitably balance generalization performance and size of the hidden layer. The latter attribute is noteworthy in that the amount of neurons plays a role in determining the resource utilization of the eventual predictor. Furthermore, random networks peculiarities have been exploited to reduce memory requirements. The two architectures outlined in the chapter address the trade-off between area utilization and latency from different perspectives. The approach that relies on a sequential computation of the neuron activations gives priority to area occupancy. As a major result, the resources utilized by the *Neuron* module does not depend on D. Conversely, the approach that relies on a fully parallel computation of the neuron activation gives priority to latency. In this configuration, the area covered by a single instance of the *Neuron* module grows as D grows. Experimental verifications proved that the proposed design strategy supports the realization of embedded classifiers both in low-end FPGA devices and low-cost devices such as CPLDs, achieving a considerable gain with respect to the previous state-of-the-artp approaches.

Chapter 5

Application: Sentiment Analysis in Text

Sentiment analysis or *Opinion Mining* is one of the suitcase problems [108] in artificial intelligence research. It can be defined as a particular application of *Data Mining* which aims to aggregate and extract emotions and feelings from different types of documents. In general, it focuses on the investigation of text, images, audio and video. It is mostly based on inference techniques which allow the aggregation of conceptual and affective information. The potential applications for this branch of *Data Mining* are countless and span interdisciplinary areas such as stock market prediction, political forecasting, social network analysis, social stream mining and human-robot interaction.

The distillation of knowledge from such a big amount of unstructured information is an extremely difficult task, as the contents of today's Web are perfectly suitable for human consumption, but remain hardly accessible to machines.

Sentiment analysis in text is a research problem that requires tackling many natural language processing (NLP) sub-tasks, including aspect extraction [109], named entity recognition [110], concept extraction [111], sarcasm detection [112], data fusion [113] and subjectivity detection.

Actually, deep neural networks made available a powerful tool to automate the process of feature extraction, which may prove challenging when dealing with complex sources of information such as images, videos or text. Such aspect becomes relevant when considering that sentiment analysis can be accomplished only by understanding the interaction between different components of a pattern. Deep learning based approaches are furtherer justified by the fact that Web 2.0 is characterized by a continuous flow of information, produced by users all over the world, enabling the use of unsupervised

techniques specifically designed for feature extraction. These streams of data are in many cases corrupted by noisy or forged information [114]. Furthermore, even when data are not forged may contain subjective information that does not provide informative contents. Filtering out this type of contents is one of the major challenges in sentiment analysis and takes the name of subjectivity detection.

Proper modeling of these phenomena using relatively simple models, as the ones introduced in the previous chapters, is challenging. On the other hand, standard solutions for subjectivity detection and sentiment analysis are computationally demanding during training and inference phases.

Given the aforementioned premises, this chapter presents a discussion about the development of algorithms with a low computational impact. In particular, it is divided into two main parts as follows:

- The first section focuses on a framework for subjectivity detection on text data based on the integration of ELM, Bayesian network, deep convolutional neural networks and recurrent neural networks. Each one of these blocks models explicitly a part of the complex mechanism that constitute subjectivity detection. As a consequence, the proposed method is more efficient in the training phase with respect to a completely deep learning based approaches. As a result, the proposed method proved appealing in presence of limited amount of training data or computational resources.
- The second part of the chapter (Sec. 5.2) focuses on the problem of developing effective text embedding. In principle, the availability of an effective embedding for textual information simplifies the entire inference problem, enabling the use of simpler models. In this thesis an investigation technique based on the combination of a new inspection algorithm for data manifold and cognitive descriptors is presented. This methodology provides engineers and data scientists with a qualitative measure of concepts distributions in a graphical format that enables a fast analysis of embedding properties. This sort of information is vital to evaluate and improve available affective computing resources.

5.1 Subjectivity Detection

Subjectivity detection is a process that aims at removing 'factual' or 'neutral' sentences, which lack sentiments, from text corpora. Such step is crucial for many sentiment-analysis technologies, which are not designed to classify 'neutral' contents. Based on

the field of application, the sentence model has to be very sensitive as users have different levels of expertise on the topic of discussion and may be from diverse economic and educational backgrounds, as well as being often separated by large geographical distances.

5.1.1 Related Works

Subjectivity detection in product reviews targets the psychology of an investor by breaking down factual information, which may imply positive or negative sentiment that is otherwise undetected by coarse grained methods that only focus on detecting explicit sentiments [115].

Another application regards monitoring response of people to different crisis situations. This is done by processing micro-blogs such as Twitter and Facebook. Here, some of the main challenges are the use of abbreviations and hashtags. Tweets may also possess dual meanings due to the potential contexts of discussion. For example in [116], the authors show that in political tweets the word 'grun' - 'green' is used for the political party 'Die Grunen' - 'The Greens', but it is also used in reference to the color green.

In [117], the authors show that subjective sentences in online forums can be identified by 'Dialog Acts' such as 'Question', 'Repeated Question', 'Clarification' etc. They also show that subjective sentences are longer than objective sentences and often contain inappropriate content such as abusive language.

Traditional sentence models extract significant n-gram features and classify them using Naïve Bayes model. For example, cloud-computing' is a bi-gram of two words frequently used together. Since, the number of such features is exponential, convolutional neural networks are being used to automatically learn them from large datasets. In [118] authors proposed the use of deep convolutional neural networks (CNNs) to extract subjectivity features in Spanish/English and the features from different languages are combined using multiple kernel learning. However, for short tweets it is difficult to compute the parameters. Further, it is often difficult to annotate long sentences by humans. For example, consider the sentence "Those digging graves for others, get engraved themselves', he [Abdullah] said while citing the example of Afghanistan." Here, there is clearly an objective frame for the writer and a direct subjective frame for Abdullah with the text anchor "said". However, it is ambiguous whether the texts anchor "citing" is objective or subjective in nature [119].

Lastly, it is possible to forecast the subjectivity in tweets using a temporal model. For example, during elections, the support for a candidate will, diffuse through a social network from nearby tweets [120].

5.1.1.1 Proposed Approach

To tackle the problem of obtain classifier with a better trade of between training time and computational cost of the inference phase, a new version of ELM called Bayesian networks extreme learning machine (BNELM) is proposed; this model is utilized to learn non-linear relationship between the hidden layer neurons. The proposed model augment the conventional SBELM by exploiting Bayesian networks (BN), which employ heuristics to determine the prior parameters of weights in the output layer [121, 122].

In practice, the resulting Bayesian Network ELM allows one to replace Laplace approximation that characterizes Bayesian theory based models with a heuristic process. The rationale behind such solution is twofold. First, Laplace approximation of SBELM is often difficult to compute, as the gradient may not exist on several noisy datasets. Second, by removing Laplace approximation one discards the corresponding non-convex optimization problem; Bayesian networks address the heuristic process by sampling a Markov chain of samples that presents better converge capabilities to the global maximum, resulting in a higher accuracy of the predictor.

The trained BNELMs are efficient compared to traditional ELMs, as they are able to prune redundant hidden neurons by learning a prior for weights. Moreover, BNELM training is more stable than the one of sparse Bayesian ELM (SBELM) because it does not need to compute the Hessian matrix of second-order derivatives that often does not exist for noisy datasets. BNELM avoid these problems by employing heuristic MCMC sampling with Gaussian Bayesian network fitness function to determine the weights between hidden neurons.

In principle, ELM is not designed to couple with datasets such as a sequence of sentences. Hence, a recurrent layer of hidden neurons is introduced to model temporal features in long sentences. Lastly, recurrent neurons can become unstable on noisy datasets hence a fuzzy classifier is used to stabilize the model and predict the output labels.

The proposed framework for subjectivity detection eventually relies on the suitable integration of CNN and BNELM. Each single element has a specific role in the process of classification. First, CNN are entitled to learn significant features from the training set. Then, the BNELM model receives as input such features. The recurrent layer supports BNELM in embedding the temporal dynamics. Indeed, as these kinds of layer are often

unstable, the BNELM employs a layer of fuzzy recurrent neurons to the specific purpose of achieving stability.

The experimental session has been designed both to verify the effectiveness of BNELM in capturing dependencies in high-dimensional data and to assess the ability of the whole framework to address learning problems characterized by a limited amount of training samples. Hence, first the Multimodal Opinion Utterances Dataset (MPQA) Gold corpus of 504 sentences manually annotated for subjectivity in Spanish is considered [123, 124]. Next, in order to evaluate the method on a multi-class problem, the Multimodal Opinion Utterances Dataset (MOUD) [125] is considered. The classification accuracy obtained using the proposed BNELM was shown to outperform the baseline consistently on both real datasets.

5.1.2 Preliminaries

This section briefly reviews the necessary concepts to the comprehension of the proposed work.

5.1.2.1 Bayesian Networks

A Bayesian network (BN) is a graphical model that represents a joint multivariate probability distribution for a set of random variables [4]. It is a directed acyclic graph that has a structure s with N nodes and a set of parameters $\boldsymbol{\theta}$, which represent the strengths of connections by conditional probabilities. Given a set of Z samples $\boldsymbol{X} = \{\boldsymbol{x}_i; \boldsymbol{x} \in \mathbb{R}^D; i = 1, .., Z\}$, the BN decomposes the likelihood of node expressions into a product of conditional probabilities by assuming the independence of non-descendant nodes, given their parents.

$$p(\boldsymbol{x}|s, \boldsymbol{\theta}) = \prod_{i=1}^{N} p(x_i|\boldsymbol{a}_i, \theta_{i,\boldsymbol{a}_i}), \tag{5.1}$$

where $p(x_i|\boldsymbol{a}_i, \theta_{i,\boldsymbol{a}_i})$ denotes the conditional probability of node expression x_i given its parent node expressions $\boldsymbol{a}_i$, and $\theta_{i,\boldsymbol{a}_i}$ denotes the maximum likelihood (ML) estimate of the conditional probabilities. Fig. 5.1 (a) illustrates the Bayesian network for a multivariate system with five nodes. Each node is a variable in the state-space of the system that can be observed or measured. The connections represent causal dependencies within a single time instance. The observed state of variable i is denoted as x_i and the regulation or conditional probability of variable i given variable j is $p(x_i|x_j)$.

A practical example of a Bayesian network representing inter-dependencies between the words of the sentence can be useful. Consider the sentence: “The escalation must end

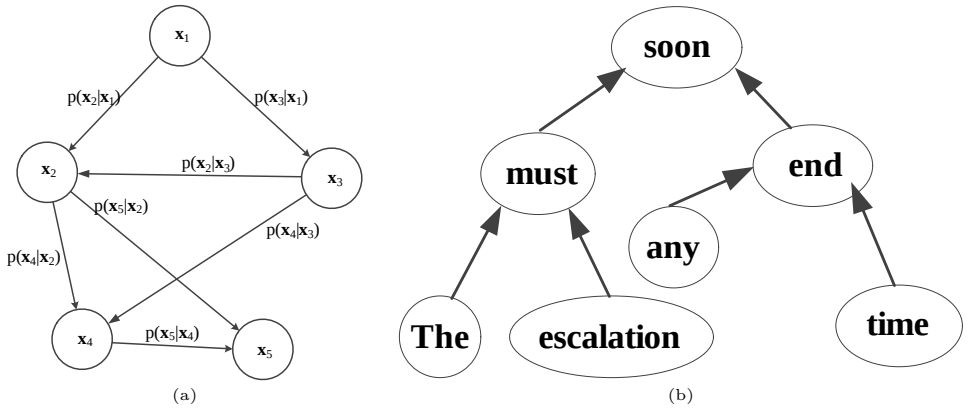


FIGURE 5.1: (a) Illustrates a Bayesian network for a multivariate system with five nodes. Each node is a variable in the state-space of the system that can be observed or measured. The connections represent causal dependencies within a single time instant. (b) Illustrates an example of a Bayesian network representing inter-dependencies between the words of the sentence 'The escalation must end any time soon'

any time soon"; the relative BN is illustrated in Fig. 5.1 (b). Once, the structure of the Bayesian network is determine heuristically using the training data, then the context that is the parents for each word can be established. For example, the context of the word 'soon' is 'must' and 'end'. Hence, the hypothesis here is that structurally related words, among all the words within the sentence, provide the best contextual information for polarity detection.

The optimal structure s^* is obtained by maximizing the posterior probability of s given the data $\boldsymbol{X}$. From Bayes theorem, the optimal structure s^* is given by

$$s^* = \arg\max_s p(s|\boldsymbol{X}) = \arg\max_s p(s) \cdot p(\boldsymbol{X}|s), \tag{5.2}$$

where $p(s)$ is the probability of the network structure and $p(\boldsymbol{X}|s)$ is the likelihood of the expression data given the network structure.

Given the set of conditional distributions with parameters $\boldsymbol{\theta} = \{\theta_{i,\boldsymbol{a}_i}\}_{i=1}^{D}$, the likelihood of the data is given by

$$p(\boldsymbol{X}|s) = \int p(\boldsymbol{X}|s,\boldsymbol{\theta}) \cdot p(\boldsymbol{\theta}|s) d\boldsymbol{\theta}, \tag{5.3}$$

To find the likelihood in (5.3), and to obtain the optimal structure as in (5.2), a standard approach consists in the use of Laplace approximation of integrals [4]. Gaussian BN assumes that the nodes are multivariate Gaussian. The parameters $\theta_{i,\boldsymbol{a}_i}$ are then defined by the mean μ and the covariance matrix Σ of size $D \times D$. The joint probability of the network can be the product of a set of conditional probability distributions is then given

by:

$$p(x_i|\boldsymbol{a}_i, \theta_{i,\boldsymbol{a}_i}) = \mathcal{N}\left(\mu_i + \sum_{j \in \boldsymbol{a}_i}(x_j - \mu_j)\beta, \Sigma_i^{'}\right), \tag{5.4}$$

where $\Sigma_i^{'} = \Sigma_i - \Sigma_{i,\boldsymbol{a}_i}\Sigma_{\boldsymbol{a}_i}^{-1}\Sigma_{i,\boldsymbol{a}_i}^{T}$ and β denotes the regression coefficient matrix, $\Sigma_i^{'}$ is the conditional variance of x_i given its parent set $\boldsymbol{a}_i$, $\Sigma_{i,\boldsymbol{a}_i}$ is the covariance between observations of x_i and the variables in $\boldsymbol{a}_i$, and $\Sigma_{\boldsymbol{a}_i}$ is the covariance matrix of $\boldsymbol{a}_i$.

5.1.2.2 Markov Chain Monte Carlo

Find the optimal structure, of a BN network is a non-convex optimization problem. Markov chain Monte Carlo (MCMC) is an optimization technique that can be employed to tackle this problem. The Metropolis-Hastings method of MCMC in particular, which associates an acceptance mechanism with newly drawn sample structures. The acceptance of a new structure s^{new} is given by the following equation:

$$\min\left\{1, \frac{p(s^{\text{new}})}{p(s)} \cdot \frac{p(s^{\text{new}}|\boldsymbol{X})}{p(s|\boldsymbol{X})} \cdot \frac{p(s^{\text{new}}|s)}{p(s|s^{\text{new}})}\right\} \tag{5.5}$$

where the Metropolis-Hastings acceptance ratio:

$$\alpha \quad = \quad \frac{p(s^{\text{new}}|\boldsymbol{X})}{p(s|\boldsymbol{X})} \cdot \frac{p(s^{\text{new}}|s)}{p(s|s^{\text{new}})}. \tag{5.6}$$

when adding an edge and the prior ratio is inverted when deleting an edge.

Sampling new structures with the use of the above-listed procedure generates a Markov chain, which converges in distribution to the approximate posterior distribution. Taking the average over sampled structures after a burn-in period, it is possible to compute the integral over parameters in (5.3). In practice, a new network structure is proposed by applying one of the elementary operations such as deleting, reversing, or adding an edge, and then discarding structures that violate the acyclic condition. The first and second term of the acceptance ratio, the ratio of likelihoods, is computed using (5.4). The third term, is obtained by

$$\frac{p(s^{\text{new}}|s)}{p(s|s^{\text{new}})} = \frac{N_n^{\text{new}}}{N_n} \tag{5.7}$$

where N_n denotes the size of the neighborhood obtained by elementary operations on structure s as well as counting the valid structures.

5.1.2.3 Sparse Bayesian ELM

Bayesian Extreme Learning Machine (BELM) is based on the use of Bayesian linear regression to optimize the weights of the output layer. The Bayes law states that the posterior distribution of model parameters is proportional to the product of the prior distribution and the likelihood:

$$p(\boldsymbol{\beta}|D) \propto p(\boldsymbol{\beta}) \cdot (D|\boldsymbol{\beta}) \tag{5.8}$$

Next, the output distribution of the model y_{new} for new input $\boldsymbol{x}_{new}$ is given by the integral of the posterior distribution of the parameters $\boldsymbol{\beta}$. Therefore, the predictive distribution for a new input is given by:

$$p(y_{new}|\boldsymbol{x}_{new}, D) = \int p(y_{new}|\boldsymbol{x}_{new}, \boldsymbol{\beta}) \cdot p(\boldsymbol{\beta}|D) d\boldsymbol{\beta} \tag{5.9}$$

where the data is assumed Gaussian, and the maximum likelihood estimate of the weights maximizes the posterior probability.

The core of sparse Bayesian extreme learning machine resides in the application of an ARD prior to the linear weight, formally

$$p(\boldsymbol{\beta}|\boldsymbol{\alpha}) = \mathcal{N}(\boldsymbol{\beta}, \boldsymbol{\alpha}) \tag{5.10}$$

The procedure of training consists in learning the parameters $\boldsymbol{\alpha}$ maximizing:

$$p(\boldsymbol{y}|\boldsymbol{\alpha}, \boldsymbol{H}) = \int p(\boldsymbol{y}|\boldsymbol{\beta}, \boldsymbol{H}) \cdot p(\boldsymbol{\beta}|\boldsymbol{\alpha}) d\boldsymbol{\beta}. \tag{5.11}$$

The integral (5.11) is intractable. However, one can address (5.11) by exploiting Laplace approximation, which involves a quadratic Taylor expansion of the log form of posterior probability. This in turn requires the computation of the Hessian matrix, which brings about second-order derivatives. The presence of noise in the dataset can indeed heavily affect this approximation.

Finally, the predictive distribution is obtained by

$$p(y_{new}|\boldsymbol{x_{new}}, \hat{\boldsymbol{\beta}}) = \frac{1}{1 + e^{-\boldsymbol{h_{new}}\hat{\boldsymbol{\beta}}}} \tag{5.12}$$

where $\hat{\boldsymbol{\beta}}$ is the Laplace's mean and $\boldsymbol{h_{new}}$ correspond to the activation of the ELM random layer with input $\boldsymbol{x}_{new}$.

5.1.3 BNELM for Subjectivity Detection

In this section, the novel Bayesian Network Extreme Learning Machine is introduced, which extends the potentialities of ELM to model sequence of sentences as dataset by exploiting the features of BNs and fuzzy recurrent NNs. In the proposed model, BNs provides an effective tool to build a network of connections among the hidden neurons of the conventional ELM configuration. Such step relies on unsupervised learning, as labels are not involved in the process. A fuzzy recurrent NN inherits the overall structure generated by the BNs to the purpose of supporting a predictor that can model also temporal features.

5.1.3.1 Bayesian Network Extreme Learning Machines

The proposed BNELM augments the standard structure of a recurrent NN to the purpose of generating a predictor that can take advantage of two main features. First, the weight matrix of connections between input nodes and hidden neurons can be learned by fully exploiting the abilities of ELMs and BNs. Second, temporal features can be suitably modeled.

In a standard recurrent NNs, the output $y(t)$ at time step t is calculated using the following equation:

$$y(t) = f(W_R \cdot y(t-1) + W \cdot \boldsymbol{x}(t)) \tag{5.13}$$

where W_R is the interconnection matrix among hidden neurons, W is the weight matrix of connections between hidden neurons and the input nodes, and f is a non-linear activation function. In BNELM, matrix W is learned by using the ELM's hidden neurons outputs $\boldsymbol{h}$ to train a Bayesian network.

Hence, the number of nodes N of the Bayesian network is equal to the number of hidden neurons. The computation of the connection matrix is achieved simply by finding the optimal structure s^* of dimension $N \times N$ defined in (5.2); accordingly, the hidden neurons outputs of the ELM become the inputs for a Gaussian Bayesian networks. Eventually, the learned structure s^* replaces the W in the recurrent layer. Algorithm 7 illustrates the complete training procedure for W with Gaussian BN fitness function and MCMC simulation.

The new structure for the recurrent layer becomes:

$$y(t) = f(W_R \cdot y(t-1) + s^* \cdot \boldsymbol{h}(t)) \tag{5.14}$$

Back propagation through time is utilized to learn W_{R}. As recurrent neurons can prove unstable on noisy datasets, a fuzzy classifier is actually exploited in this work to stabilize the model. Therefore, $\mathrm{y}(t)$ eventually becomes $\boldsymbol{y}(t)$ by introducing fuzzy membership functions.

The BNELM model has two main advantages with respect to SBELM. First, training a SBELM involves the computation of second-order derivatives. The performance of this procedure is highly sensible to noisy datasets. In BNELM training, this problem is overcoming by employing heuristic MCMC. Second, SBELM involves non-convex optimization problem; BNs, though, address the heuristic process by sampling a Markov chain of samples that theoretically always converge to the global maximum at the equilibrium, resulting in a higher accuracy of the predictor.

Moreover, one should consider that determining the number of optimal hidden neurons is one of the biggest challenges in ELM. The MCMC algorithm will only consider edges with frequency above a threshold in all samples when determining the optimal structure s^*. Hence, neurons with no edges are removed from the structure. In this way, BN is able to determine the optimal number of hidden neurons, thus pruning redundant neurons.

Algorithm 7 Bayesian Network Extreme Learning Machine training

Input

- a labeled training set $\{(\boldsymbol{x}, \boldsymbol{y})_i; \boldsymbol{x} \in \mathbb{R}^N; \boldsymbol{y} \in \mathbb{R}^P; i = 1, .., Z\}$;
- number of neurons N
- acceptance rate α

0. **Initialize**
   Randomly initialize weights of the input hidden layer, with N nodes
   Build a BN with N nodes, one for each hidden layer activation $\boldsymbol{h}$
   Initialize topology of $s = s_0$
1. **Iterative training**
   repeat
     Generate s^{new} by elementary operations
     Find N^{new} and N^{old} corresponding to s^{new} and s
     Find acceptance ratio α given by (5.6)
     if $\alpha \geq 1$ **then**
       $s = s^{new}$
     else
       if $rand[0, 1] \geq \alpha$ **then**
         $s = s^{new}$
       end if
     end if
   until convergence
   $s^* \cdot \boldsymbol{h}$ is used to train the final fuzzy recurrent classifier

5.1.3.2 A Framework for Subjectivity Detection

The BNELM is a suitable tool to support the development of a framework for subjectivity detection. Figure 5.2 schematizes the framework by proposing the whole flowchart.

The BNELM receives as input a vector $\tilde{\boldsymbol{x}}$, which is the outcome of a pre-processing step involving a deep CNN model. First, a sentence is transformed into word vector representation $\boldsymbol{X}$ of dimensions $L \times d$, where L is the maximum number of words in a sentence, and d is the number of Google word vectors used. The transformed data then feed the deep CNN model, which automatically perform dimensionality reduction. Deep CNN are being used extensively to extract patterns automatically from large datasets such as Twitter. Such a model looks for highly activated k-grams in a CNN as our pattern set. For example, 'cloud-computing' is a bi-gram of two words 'cloud' and 'computing'. Details of such an implementation can be found in [118]. However deep models are extremely slow. Hence, here the activations at the penultimate layer of the deep CNN is used as a new training data for a fast ELM model. The first CNN hidden layer contains kernels of size $k \times d$ to learn k-gram patterns. There are several layers of kernels and an output layer of n_d sentiment labels namely 'positive', 'negative' and 'neutral'. The features learned by deep CNN are expressed in penultimate layer and can be used as input $\tilde{\boldsymbol{x}}$ to the BNELM model.

The BNELM in practice is entitled to learn the context of n-grams elaborated by the CNN. Since BNELM embeds a recurrent layer of hidden neurons, the framework is able to model temporal features in long product reviews. In Figure 5.2, the fuzzy membership function is designed to tackle a three-class problem. The complete training procedure is proposed in algorithm 8.

A detailed description of the framework is proposed in Figure 5.3. Starting from the bottom, the processing is organized as follows:

- The first two layers starting from the bottom of the image correspond to the Bayesian network used in the pre-processing step.
- The most important $n-$gram features are extracted from sentences using deep CNN. In the figure, CNN corresponds to the third layer. The bold lines identify the size of the kernel window, which has been set to 4.
- The extracted features are then fed into the random layer of the BNELM; in this figure, the links are represented with dashed lines. The output weights of the random layer have been learned using a Bayesian network.

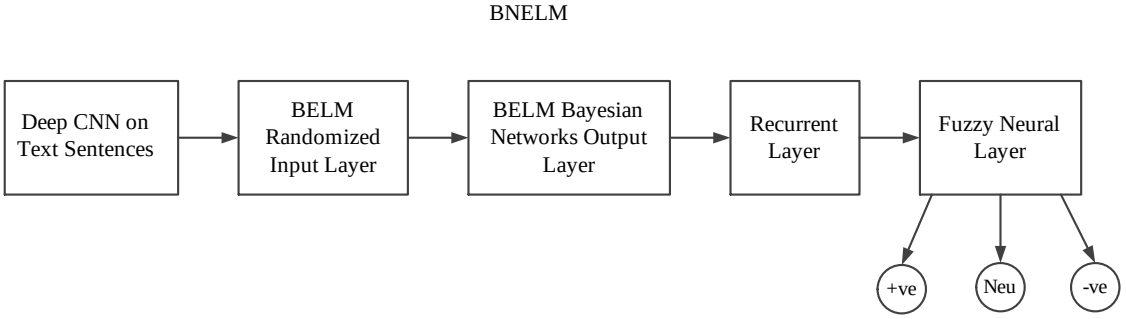


FIGURE 5.2: Illustrates the flow chart for BNELM framework. First, a conventional deep CNN process the sentences. The extracted features are fed into an ELM classifier, where the output layer weights are determined heuristically using Bayesian Networks. Features learned are further evolved by BNELM using a Fuzzy Recurrent neural network. The output layer has three nodes for classifying positive, negative, or neutral sentences.

- The ELM output is used to train a layer of recurrent neurons with feedback connections, marked as "Inter-connected recurrent neurons" in the graph.
- The two fuzzy membership functions are used to facilitate stable convergence of the model.
- The top layer represents the output layer of the network. It has three neurons; one per class.

Algorithm 8 Subjectivity detection framework training

Input

A labeled training set $\{(\boldsymbol{x}, y)_i; i = 1, .., Z\}$ where each $\boldsymbol{x}$ is a sentence of length L and labels $y \in \{Pos, Neu, Neg\}$

0. **Feature extraction**

   Transform each sentence in vector representation with word vector dimension d
   Construct deep CNN with visible layer as a 2-d vector of $L \times d$ input features
   Construct hidden layer with kernels of size $k \times d$ to learn k-gram patterns
   Construct several hidden layers with kernels and output layer with n_d neurons

1. **ELM training**

   The features learned by deep CNN are expressed in penultimate layer and can be used as input layer to BNELM model

5.1.3.3 Computational Complexity

The computational complexity of a single training epoch for the l^{th} convolutional layer of a CNN is given by $\boldsymbol{O}(n_{l-1}.s_l^2.n_l.m_l^2)$, where n_{l-1} and n_l are, respectively, the number of input and output feature maps; $s_l = n_x^{l-1} \times n_y^{l-1}$ and $m_l = n_x^l \times n_y^l$ are, respectively,

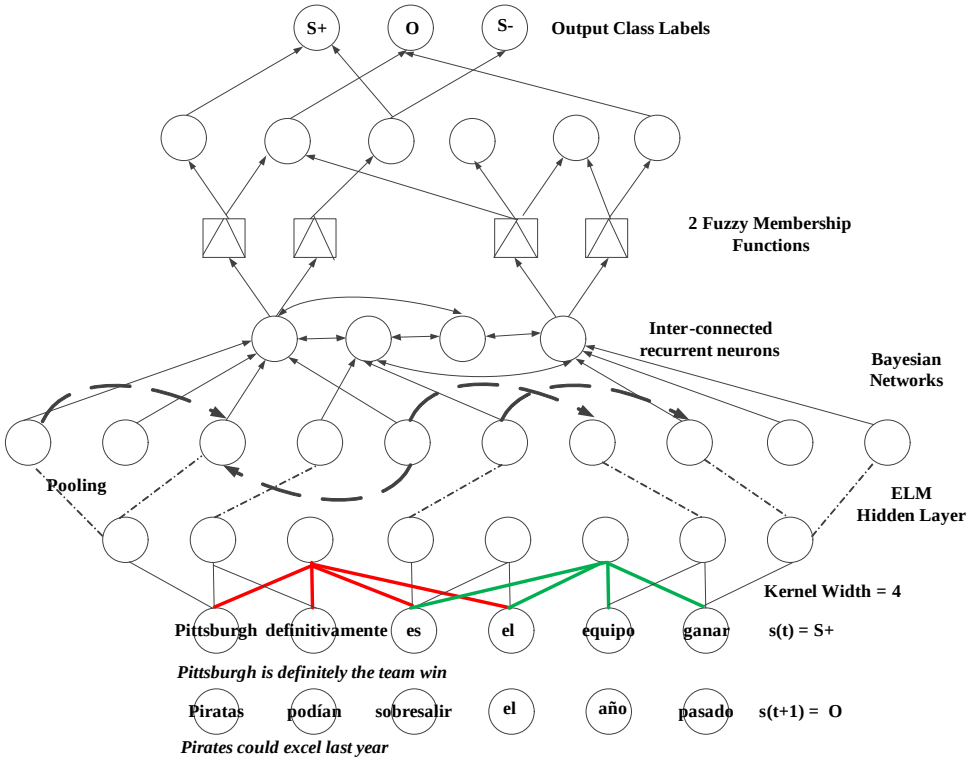


FIGURE 5.3: Illustrates the state space of a BNELM for a subjective sentence in online forums. Features are extracted from Spanish sentences using deep CNN. The bold lines correspond to kernels. The extracted features are then used to train a Bayesian ELM, where output layer weights are determined using Bayesian networks. The bold dashed arcs correspond to causal edges predicted by a Bayesian network. The ELM output is subsequently used to train a layer of recurrent neurons with feedback connections. Lastly, the framework embeds a layer of fuzzy neurons with 2 membership functions in order to achieve stable convergence of the model.

the dimensions of the input and output feature maps. This computational cost clearly characterizes the computational complexity of the overall framework.

In contrast, the computational complexity of a layer of recurrent hidden neurons is much lower being $\boldsymbol{O}(2 \times N^2)$, where N is the number of neurons and a single time delay is considered. Similarly, the complexity of the neuro-fuzzy classifier with 2 membership functions is also very small being $\boldsymbol{O}(\sum_{i=1}^{n_{l-1}} 2n_l + \sum_{i=1}^{n_{l-1}} 2)$ [126].

In general, training a CNN as a classifier is indeed time consuming due to the huge number of training iterations required. In the proposed method, however, the CNN is utilized as a feature extractor; eventually, such features are feed into a low-dimensional Bayesian ELM model. As a result, in this case, the CNN only involves a reduced number of epochs; this in turn means a significant reduction of the computational cost.

Lastly, the present framework is designed to exploit a small number of features learned by deep learning. Thus, the computational cost of the MCMC with BN fitness function

also decreases. It is worth noting that model obtained after the training process is sparse; this is a major difference with the model one would obtain by exploiting a standard ELM model. As a consequence, the computational performance of the eventual predictor is better.

5.1.4 Experiments and Results

There are two aims of the experiments: first, to evaluate the generalization performance of the proposed BNELM, second, to provide a comparison between the proposed method and two alternative models, that is, a classifier based on the standard regularized ELM [127] and a classifier based on a sparse Bayesian ELM [128]. Three different benchmarks have been used in the experimental evaluation: MPQA Gold corpus [123, 124], Utterance-Level Multimodal Sentiment Analysis [129] and TASS 2015 Corpus. Both datasets involve sentences expressed in Spanish. The rationale behind this setup is that present research is geared towards assessing the proposed methods ability to deal with non-English language documents, proving the ability of the model to tackle problems with a limited source of labeled data. In all the experiments, standard model-selection procedures support the setup of the regularization parameter λ (ELM). The following settings have been used:

$$\lambda \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 1, 10^{1}, 10^{2}, 10^{3}, 10^{4}, 10^{5}, 10^{6}\}$$

The overall experimental setup involved the mixed use of the Perl programming language and Python during preprocessing stage, while training and evaluation were developed using together Matlab and Python softwares.

5.1.4.1 Preprocessing

A pre-processing stage has been applied to all the datasets involved in the experimental session. The first step consisted of removing the top 50 stop words and punctuation marks from the sentences [1]. Next, a part of speech (POS) tagger was used to determine the part-of-speech for each word in a sentence. Words may have different subjectivity levels when used in different forms such as 'noun' or 'verb', hence POS tagging was applied to all the Spanish training sentences.

After POS tagging, subjectivity clue words were identified. The subjectivity clues dataset [130] contains a list of more than 8,000 clues identified both manually as well as automatically, using both annotated, and un-annotated data. As this dataset includes

[1] http://www.ranks.nl/stopwords/

only English words, the corresponding Spanish list was created using the Bing translator, API. For each clue word, the number of occurrences in the dataset was computed. Eventually, the top 50 clue words with highest occurrences in the subjective sentences were considered [2]. Each sentence was then transformed to a binary feature vector of length 50, where the presence of a clue word is denoted as '1' and an absence is denoted as '0'.

The resulting binary matrix 'clue words versus sentences' has been processed as a time series. Thus, the sentences have been used as input for a Gaussian Bayesian networks. The maximum likelihood (ML) probabilities of each word, given up-to three parent words and up-to two time points delay, was also computed. Such sub-structures are referred to as network motifs. The top 20% of motifs with the highest ML were exploited to select the pre-training sentences for a deep CNN, done by simply selecting the sentences containing these motifs.

The deep CNN was employed to extract features in the form of 3-grams and 4-grams in each language separately. It was configured as follows: three hidden layers with 100 neurons each, kernels of size 3, 4 and 5, respectively, and one logistic layer with 300 neurons. The output layer included two neurons for each class of sentiments. The 300 feature outputs of the deep CNN -from both languages- were used to train the BNELM with an additional fuzzy recurrent layer of 10 hidden neurons and up to 2-time point delays.

5.1.4.2 MPQA Gold Corpus

The MPQA Gold corpus is a collection of 504 sentences manually annotated for subjectivity in Spanish. The annotation resulted in 273 subjective and 231 objective sentences [131]; the corpus includes sentences of an uncertain nature that were assigned to a definite class after assessment by multiple annotators. MPQA Gold is a popular benchmark, which can be used to evaluate the robustness of the proposed framework when a small training set is involved. The sentences were eventually machine translated into English to obtain the final dataset, which after pre-processing lay in a 20-dimensional space. A 5-fold cross validation has been used to estimate the accuracy of the trained classifier when applied to new sentences, i.e., sentences not included in the training set. The performance of the three predictors (BNELM, ELM, and SBELM) was assessed by using the average value of the accuracy computed over 10 runs, i.e., 10 different randomizations of the mapping layer.

Figures 5.4 and 5.5 provide the outcomes of the experiments. Figure 5.4 assesses the performances of ELM, SBELM, and BNELM for four different sizes of the mapping layer:

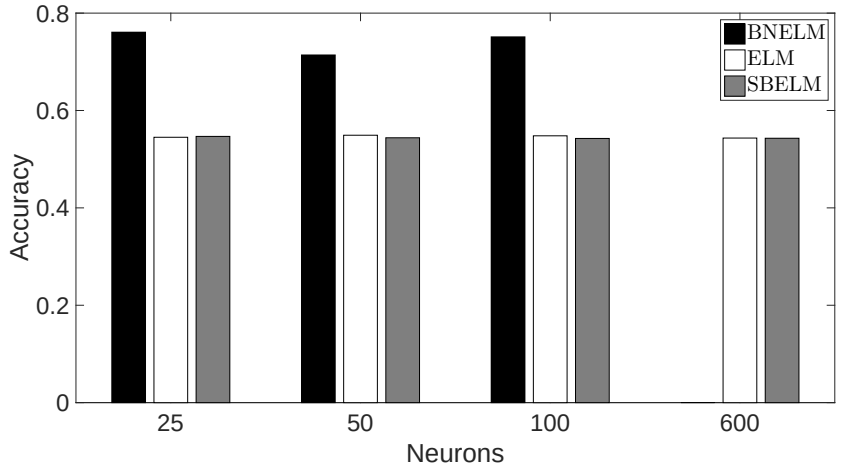


FIGURE 5.4: Accuracy of experiments involving mpqa gold corpus dataset; Average accuracy of the three different classifiers;

$L = \{25; 50; 100; 600\}$, note that SBELM has not been tested for a random hidden layer of 600 neurons, due to the expensive training phase. All the experiments were run using a fixed value for the number of iterations, $nItr$, in the MCMC procedure. In figure 5.4, the x axis gives L, while the y axis gives the classification accuracy (expressed as the percentage over the size of the test set). The bar graph compares the performance of the standard ELM (white bar) with the performance of the SBELM (grey bar) and the performance of BNELM (black bar). On an overall basis, the graphs clearly show that the BNELM can improve over standard ELM and SBELM in terms of classification performance. Figure 5.5 analyses the performance of the BNELM for different values of the parameter $nItr$. Here, the x axis gives number of iterations $nItr$, while the y axis gives the classification accuracy. The experiments refer to a configuration with $L = 50$; the number of iterations were allowed to take the following values: $nItr = \{25; 50; 100\}$. The graph shows that the accuracy of the classifier reaches a maximum when the number of iterations is 25. Nonetheless, it is interesting to note that a good accuracy can be obtained with all the proposed values of $nItr$.

5.1.4.3 Multimodal Opinion Utterances Dataset

The dataset 'Multimodal Opinion Utterances Dataset' consists of 498 short video fragments where a person utters one sentence. The items are manually tagged for sentiment polarity, which can be positive, negative, or neutral. The videos are in MP4 format with a resolution of 360 $\times$ 480 pixels; the duration of the clips is about 5 seconds on average. About 80% of the clips involve female speakers. The transcripts of the videos were used as a dataset. After pre-processing, the patterns lay in a 20-dimensional space.

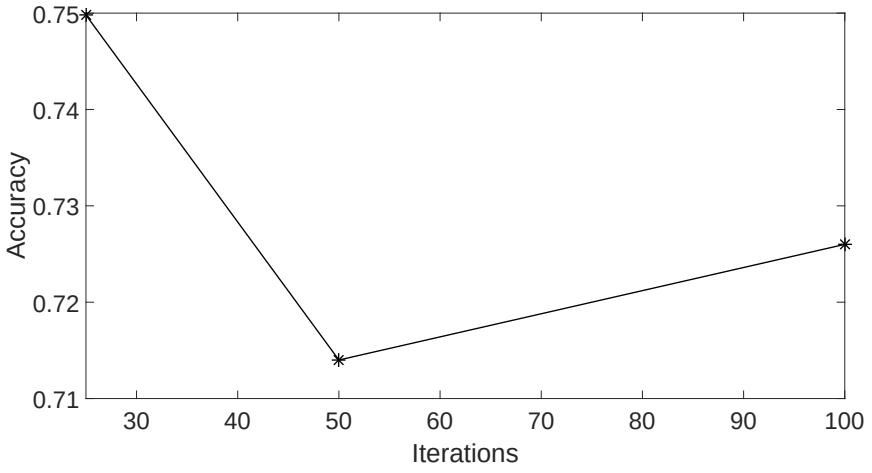


FIGURE 5.5: Accuracy of experiments involving mpqa gold corpus dataset; Average accuracy of the gmm-ELM for different values of the parameters $nItr$;

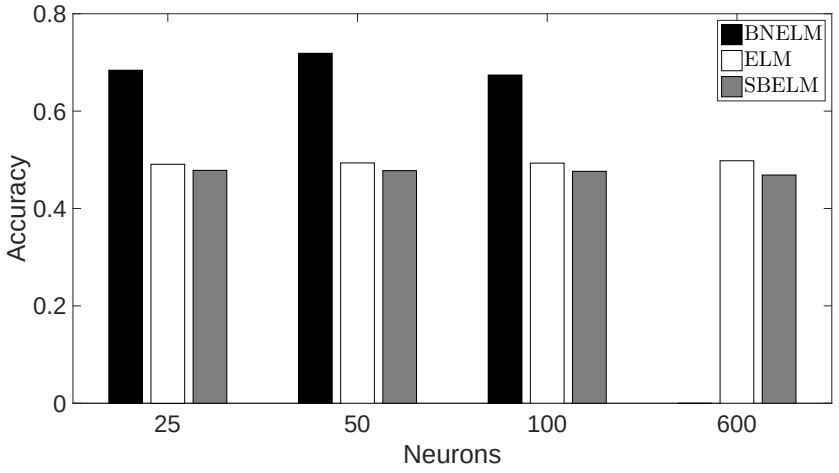


FIGURE 5.6: Accuracy of experiments involving Multimodal Opinion Utterances Dataset; average accuracy of the three different classifiers;

Similar to the MPQA experiment, a 5-fold cross validation has been used to estimate the performance of the trained classifier. Figures 5.6 and 5.7 provide the outcomes of the experiments by adopting the format of Fig. 5.4 and Fig. 5.5, respectively. The experiments showed in Fig. 5.6 involved the following settings: $L = \{25; 50; 100; 600\}$; $nItr = 50$. The settings of the experiments showed in Fig. 5.7 are: $L = 50$; $nItr = \{25; 50; 100\}$. In general, these results confirm the tendency identified with the MPQA gold corpus dataset. This in turn proves that BNELM can indeed serve as an effective tool to deal with sentiment analysis.

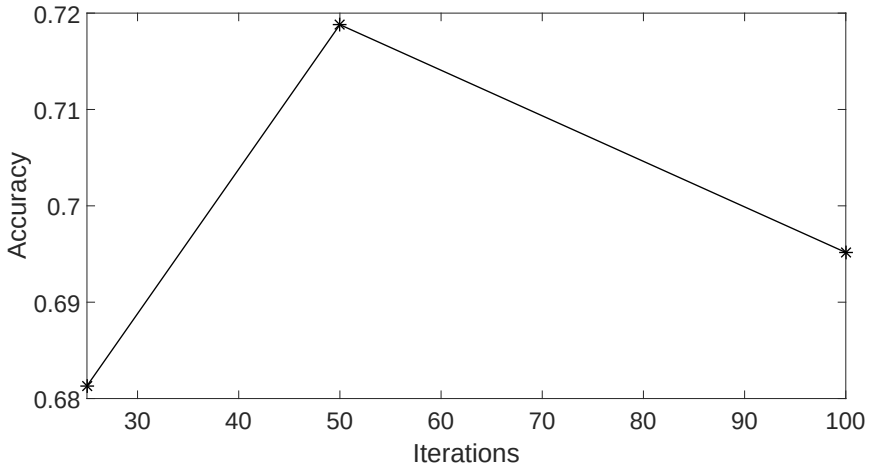


FIGURE 5.7: Accuracy of experiments involving Multimodal Opinion Utterances Dataset; Average accuracy of the gmm-ELM for different values of the parameters $nItr$;

TABLE 5.1: Accuracy by different models for classifying sentences in a document as Positive(Subjective), Negative(Subjective), Neutral(Objective) or None in TASS dataset.

	CNN [134]	LYS [133]	LIF [132]	BNELM
TASS 2015	0.66	0.637	0.692	**0.89**

5.1.4.4 Sentiment Classification on the TASS 2015 Corpus

In order to evaluate the model on a noisy dataset, we consider the four class TASS corpora of Spanish tweets [132]. Each tweet belongs to one of the four categories: *positive*, *neutral*, *negative*, or *without opinion*. In this test, a training set of 7219 tweets and the test set of 1000 tweets are used.

Table 5.1 shows accuracy by different models for classifying sentences in a document as Positive (Subjective), Negative (Subjective), Neutral (mixed) or None in TASS test dataset. A simple CNN model for sentences learns features of two or three words using sliding window kernels. We also compare our approach with different models evaluated at the TASS workshop (see the overview paper [132] for a detailed description of all approaches).

In LYS [133], the authors used classical logistic regression with linguistic features. Their approach was limited as they relied heavily on polarity lexicons that are not available in Spanish, instead extreme learning machines is employed to automatically learn features from both English and Spanish. Further, we use Bayesian heuristics to determine parameters for a large set of noisy tweets and hence are able to outperform the baselines by over 15% in accuracy.

5.1.5 Concluding Remarks

Subjectivity detection represents a challenging task for sentiment-analysis tools. This research introduces a novel architecture for machine learning that is designed to effectively support that task. The proposed BNELM augments the standard structure of a recurrent NN to the purpose of generating a predictor that can take advantage of the fruitful properties of ELM and Bayesian Networks.

The BNELM architecture has two main advantages with respect to SBELM. First, training BNELM does not involve the computation of second-order derivatives. Second, BNELM tackles the optimization problem by exploiting a heuristic procedure that relies on MCMC. As a result, the optimization process tends to converge to the global maximum at the equilibrium, resulting in a higher accuracy of the predictor. Moreover, the use of BNs inherently leads to a model that is able to determine the optimal number of hidden neurons, thus pruning redundant neurons.

The eventual framework for subjectivity detection relies on the suitable integration of CNNs and BNELM. First, CNN are entitled to learn significant features from the training set. Then, such features feed the BNELM model. Experimental results confirmed the effectiveness of the proposed method.

5.2 Graphical Exploration of Text Embedding for Sentiment Analysis

As introduced in previous section 5.1, the quality of the whole investigation process relies on a suitable conversion of the textual information in a numerical format, generally called embedding. The availability of effective embeddings capable of encode complex interaction between words simplifies the inference problem enabling the use of simpler learning paradigms. This aspect becomes crucial in the field of embedded systems where memory constraints and computational power limits the use of complex models. For this reason, the development of an embedding is a primal challenge in all the branches of text mining [33, 135–138]. An embedding can be formalized as a transformation $F : T \to \mathbb{R}^d$ where $d \in \mathbb{N}^+$ and T is the set of admissible fragments of text.

It is possible to individuate two macro categories of embedding, namely general purpose and task specific. The first group consists of remapping methodologies based on mere statistical properties of the text; the second group instead contains the methodologies suited to build a numerical spaces based on task specific constraints. Astonishing results have been achieved using "general purpose embedding" combined with deep networks

methodologies and standard approaches can be used to tailor this representations to specific tasks using external resources [139–141]. Even if it is always possible to generate a task specific embedding from general purpose ones, it should be observed that this operation is computationally and data demanding. For this reason, in most of the real-world scenarios, the use of available resources is preferred [137, 142]. Different available sentiment lexicons such as the multi-perspective question answering (MPQA) corpus [143], NRC [144], and SenticNet [145] provides directly sentiment polarities of words; even if focused, this scalar information is limited for the task of model the complex interactions between terms and sentiments. Li et al. [146], recently, proposed a technique for the development of a more refined sentiment lexicon based on the GloVe embedding [142] evolved with the use of external resources. Another example of sentiment oriented embedding, can be found in [147] where the authors proposed the use of the Harvard psychological dictionary and Loughran-McDonald financial sentiment dictionary to build a sentiment space. Concept-based approaches have been proved to be more effective than word based ones because sentiments are in most of the cases related to the combination of many terms. AffectiveSpace [1] is a 100 dimensional embedding of concepts, built respectively using the SVD and the random projections on the top of AffectNet, a matrix of affective commonsense knowledge.

Measuring embedding quality is an ambiguous and task specific work that requires to tackle various problems; understanding the topology and properties of data distributions in these high dimensional spaces [148] is essential to obtain improvements in the field. One of the most evident issue is related to visualization of high dimensional spaces [149], typically addressed via dimensionality reduction techniques [150–154]. Another issue is identifying a correct metric between concepts in the embedding space; standard methodologies consider Euclidean distance or normalized scalar product, usually paired with clustering algorithms, to define a concept of similarity between data. Finally, consistence of data disposition and psychological theories is almost an unexplored field, in particular, no works are available about the consistency of remapped data distributions and physiological models.

Recently, [155] proposed a novel algorithm, a regularized k-means (RKM) suited to find so called, principals path in data space. The core idea is finding a discrete set of points that describe the transition between two samples guided by the underlying distribution. The ability of providing information about data morphologies in high dimensional spaces is particularly appealing for the task of analyzing embedding quality because transitions unveil details related not only to local data distribution, but embeds information about the global displacement of the data.

This section presents a new tool based on the combination of the RKM algorithm and a set of descriptors inherited by psychology. This instrument enables a cognitive analysis of concept displacement in the high dimensional spaces. A peculiar feature of this tool is that it enables a qualitative study of data distribution morphology by means of manifolds, using task specific descriptors that make possible a direct analysis from a human user. A second outcome of the proposed work consist on a qualitative characterization of AffectiveSpace where the user is provided with a series of cognitive descriptors about the data distribution.

The main outcome of this work is the development of an analytic instrument for cognitive reasoning in high dimensional spaces derived by concepts embedding. The proposed method is then tested on AffectiveSpace, providing a graphical representation of the distribution of concepts in this space; results of this analysis confirm qualitatively the consistency of AffectiveSpace and Plutchik's theories [156]. The hope is that the proposed analytical tool will help engineers and data analysts in the development of embeddings that are increasingly able to encode all the characteristic related to Sentiments in a numeric format, boosting the possibility of implement efficient classifiers in resource constrained environments.

The rest of the work is organized as follows: the first subsection reviews the previous works that constitute the basis of the presented method; the subsequent subsection instead presents the details about the proposed framework. Subsections 5.2.3 and 5.2.4 present respectively the experimental campaign and the results. Finally, the last subsection briefly summarizes the conclusions.

5.2.1 Preliminaries

The concepts necessary to understand the rest of the work are presented. Firstly, the selected psychological model is introduced, followed by a brief explanation of the mathematical model for the inspection of data manifolds in high dimensional spaces.

5.2.1.1 Hourglass Model

Among the available models suited for the description of sentiments, the so called Hourglass of Emotions [157] has been considered. It represents the core element of many studies of sentic computing. It is founded on Plutchik's studies on human emotions and consists in a reorganization of sentiments around four independent dimensions whose different levels of activation make up the total emotional state of the mind. The Hourglass of Emotions depicted in Figure 5.8, is based on the idea that different independent

resources master the emotional state. Different conditions results from turning some set of these resources on and turning another set of them off. In each of such configuration the behavior of the brain changes: the state of 'anger', for example, appears to privilege a set of resources that support an immediate reaction to external stimulus while also suppressing some other resources that usually make us act prudently. The Hourglass shape derives by the fact that an emotion can be identified only if it is strong enough, i.e., a person cannot feel a specific emotion like 'fear' or 'amazement' without that emotion being reasonably strong. The model does not consist of a classification of affective states in basic emotional categories, but models the affective state using four concomitant but independent dimensions, characterized by six levels of activation, which determine the intensity of the expressed/perceived emotion as a float $\in$ [-1,+1]. The six activation level for each of the affective dimensions provides a labelled set of 24 basic emotions in a way that allows the model to specify the affective information associated with text both in a dimensional and in a discrete form. The numerical representation of concepts can be retrieved using [145].

5.2.1.2 Principal Path in Data Space by RKM

Regularized K-Means [155] (RKM) is an algorithm for the selection of smooth paths in data space founded on the idea that, given a set of points and two reference points in

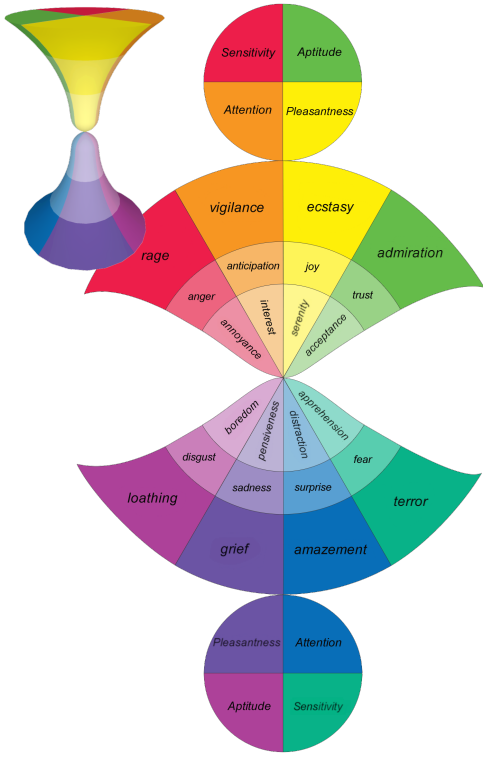


FIGURE 5.8: The hourglass of emotions [1]

a vector space $X \in \mathbb{R}^d$ where $d \in N^+$, it is possible to find a morphism between them exploiting the information provided by the available data. This morphism is described as a discrete path, composed of a set of prototypes selected based on the data manifolds.

Formally speaking, consider a set of points $\boldsymbol{X} = \{\boldsymbol{x}_j \in \mathbb{R}^d\}, j = 1, ..., N$ and two points $\boldsymbol{w}_0$ and $\boldsymbol{w}_{N_c} \in \mathbb{R}^d$. The path connecting the two points $\boldsymbol{w}_0$ and $\boldsymbol{w}_{N_c+1}$ is described as an ordered set $\boldsymbol{W}$ of N_c prototypes $\boldsymbol{w} \in \mathbb{R}^d$. The path is found by minimizing standard K-means cost function with the addition of a regularization term that considers the distance between ordered centroids. The cost function can be formalized as:

$$\min_{\boldsymbol{W}} \frac{\gamma}{2} \sum_{i=1}^{N} \sum_{j=1}^{N_c} \|\boldsymbol{x}_i - \boldsymbol{w}_j\|^2 \delta(u_i, j) + \frac{\lambda}{2} \sum_{i=0}^{N_c} \|\boldsymbol{w}_{i+1} - \boldsymbol{w}_i\|^2 \tag{5.15}$$

where u_i is the datum cluster. The novel cost function is composed of two terms weighted by the hyper-parameters γ and λ:

$$\Omega(\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{X}, \gamma, \lambda) = \gamma \Omega_X(\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{X}) + \lambda \Omega_W(\boldsymbol{W}) \tag{5.16}$$

The first term coincides with the standard K-means cost function while the second one induces a path topology due the centroids ordering and controls the level of smoothness of the path. Figure 5.9 provides a graphical example of the algorithm behavior in a two dimensional space, for different values of the regularization hyper-parameters; in the graph, data are represented as blue dots and centroids as crosses; blue line refers to a configuration in which the first cost function term is prominent, the green to a configuration where the second term of the cost function is preponderant, instead the red refers to a configurations with a right trade-off between the two.

The minimum of the cost function is obtained through the use of an expectation maximization algorithm [158]. The procedure can be summarized as:

- *E-step*: consider iteration t of the optimization process and $\boldsymbol{W}_t$ the set of prototypes. Minimizing $\Omega(\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{X}, \gamma, \lambda)$ respect to u is equivalent to minimize $\Omega_X(\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{X})$ that is the standard K-means cost function.

$$u_{i,t+1} \leftarrow \arg\min_j \|\boldsymbol{x}_i - \boldsymbol{w}_{j,t}\|^2 \tag{5.17}$$

- *M-step*: based on the newly computed $\boldsymbol{u}_{t+1}$, minimization of $\Omega(\boldsymbol{W}, \boldsymbol{u}, \boldsymbol{X}, \gamma, \lambda)$ respect to $\boldsymbol{W}$ is performed. An approximated closed form solution can be computed. Details about it can be found in the original paper [155].

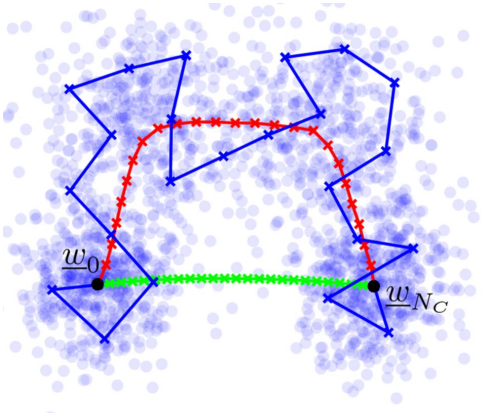


FIGURE 5.9: Example of hyper parameters influence in the shape of the path where data are represented as blue dots and centroids as crosses; blue line refers to a configuration in which the first cost function (5.15) term is prominent, the green to a configuration where the second term of the cost function is preponderant, instead the red refers to a configurations with a right trade-off between the two.

This new cost function is coupled with the framework of Bayesian evidence maximization for the selection of hyper-parameters. Interestingly, the choice of the regularization parameter induced by the evidence framework has the further advantage of virtually removing the other degree of freedom, namely the number of clusters, finding almost the same manifold even varying the number of clusters, reducing that parameter to a mere discretization coefficient of the path.

Algorithmically, RKM can be seen as a function of two extreme points $\boldsymbol{w_0}$ and $\boldsymbol{w_{N_c+1}}$, a set of points in the same space $\boldsymbol{X}$ and a discretization parameter N_c, with an output consisting of an order set of point $\boldsymbol{W}$:

$$\boldsymbol{W} = RKM(\boldsymbol{w_0}, \boldsymbol{w_{N_c+1}}, \boldsymbol{X}, N_c); \tag{5.18}$$

5.2.2 Proposed Methodology

This subsection aims at providing a tool to investigate the structure of high dimensional spaces induced by embedding from a cognitive point of view. To achieve this goal, one needs a methodology that 1) inspects the geometrical disposition of data (i.e., concepts) in $\mathbb{R}^d$, and 2) links this information to cognitive descriptors. Ideally, the tool should provide this information in the most intuitive and effective way but, at the same time, the information should be as concise as possible.

The design of this tool can be divided in two parts. First, subsection 5.2.2.1 explains how the RKM algorithm can be used to inspect concepts distribution in a high dimensional

space. Second, subsection 5.2.2.2 introduces the procedure that links the topology to the psychological descriptors proposed in [1].

5.2.2.1 Space Exploration

The RKM algorithm provides a suitable tool to extract information about the topology of concepts distribution in a high dimensional space. In this regard, let C be a set of N concepts and let $\{\boldsymbol{x}_1, .., \boldsymbol{x}_N\} \in \mathbb{R}^d$ their projections induced by embedding F. Besides, let be l_{start}, $l_{end} \in C$ two generic concepts that will correspond to the two extremes of the path under analysis. Accordingly, RKM can be exploited to identify the path that connects l_{start} with l_{end} in $\mathbb{R}^d$. Thus, the algorithm output is the list of intermediate concepts that characterize the transition induced by the data distribution.

For the sake of clarity, it is useful to consider again Figure 5.9. In this case, each point corresponds to the $2D$ projection $\boldsymbol{x}_i$ of a concept $c_i \in C$; $\boldsymbol{w}_0$ and $\boldsymbol{w}_{N_c+1}$ are respectively the projections of l_{start} and l_{end}. The path selected by the algorithm would represent the most natural transformation between the two concepts; each of the prototypes identifies an intermediate step of this transformation. It is worth noting that the prototypes $\boldsymbol{w}_j \in \mathbb{R}^d$ placed by RKM are not, in general, coincident with known concepts in C. In principle, however, each prototype represents the projection of an unknown concept.

Algorithm 9 formalizes the procedure adopted to move between the two spaces (textual and numerical) connected by the embedding. Given an embedding F, the projections of two concepts belonging to C are obtained: eventually, the two points became $\boldsymbol{w}_0$ and $\boldsymbol{w}_{N_c+1}$. The path between the two, provided by the distribution of the concepts in the embedding space, is found using RKM algorithm. The set of N_c prototypes in $\mathbb{R}^d$ is then provided as output; indeed, the generic prototype $\boldsymbol{w}_j$ does not correspond to the projection of a known concept $\boldsymbol{x}_i$. Thus, each single prototype is linked to a known concept by using the closest concept $\boldsymbol{x}_i$ in terms of Euclidean distance in the remapped space; such metric is indeed coherent with the metric used by RKM algorithm.

Algorithm 9 can be applied to any embedding F. Besides, by defining multiple pairs $\{l_{start}, l_{end}\}$ one can explore cogently the underlying space of concepts. In this work, the landmarks are provided by the concepts corresponding to the activation of the 4 dimensions individuated by the hourglass model, i.e., the concepts that can be used to obtain a discrete quantization of the space. This setup implies the reasonable assumption that the 24 archetype concepts are included in the embedding under analysis.

The strategy adopted in this research is based on the independent analysis of the 4 principal dimensions. Indeed, the goal is to characterize the transitions between the

adjacent activations of a given dimension. As an example, let *Sensitivity* the dimension to be analyzed. Then, Algorithm 9 is run by using as pairs $\{l_{start}, l_{end}\}$, respectively, {'rage','anger'}, {'anger','annoyance'}, {'annoyance','apprehension'}, {'apprehension', 'fear'}, and {'fear','terror'}. The rationale behind such strategy is that the four dimensions are expected to be independent from each other; then, in principle, when moving along one of these dimensions the activations of the remaining three should not vary. By linking the 5 paths that characterize a single dimension one gets the transition from the concept that corresponds to the highest level of activation to the concept that corresponds to the lowest activation level.

Overall, this strategy leads to 20 paths (5 paths for each dimension). As a result, one explores the space moving through the manifolds of each dimension, using as references points the concepts relative to the activations of the hourglass model. The eventual outcome, actually, is not only the information about the morphism between two extremes of a path. In addition, one also unveils a set of descriptors (i.e., concepts) for the underlying data manifolds.

A perturbation scheme has been applied to obtain a more complete set of manifold's descriptors. Given a pair of concepts l_{start},l_{end}, Algorithm 9 has been applied Q times; at each run, the set of concepts C was sub-sampled, obtaining a new set of subspaces $\tilde{C}_i \subset C$, with $i = 1, .., Q$. At the end of the procedure, thus, Q different paths describing the transitions induced by different sub-sampling of the space were collected. On the one hand, it is true that a similar mechanism could be implemented by using different

Algorithm 9 Mechanism of space exploration

Input

- Embedding F with domain C and relative projections $X \in \mathbb{R}^{N \times d}$
- Pair of landmarks concepts l_{start} and $l_{end} \in cL$
- Number of prototypes N_c

Output

List of concepts representing transitions cL

Procedure

$w_0 = F(l_{start})$
$w_{N_c+1} = F(l_{end})$
$path = RKM(w_0, w_{N_c+1}, X)$
for $j \leftarrow 1$ **to** $length(path)$ **do**
  $L(j) = \arg\min_{\boldsymbol{x} \in \boldsymbol{X}} \|\boldsymbol{path}(j) - \boldsymbol{x}\|^2$
  $cL(j) = F(Lj)^{-1}$
end for

return cL

initializations for the set of prototypes, as the minimum problem is not convex. However, the sub-sampling procedure can induce more variance in the set of solutions.

5.2.2.2 Descriptors

The information extracted by "space exploration" phase consists of Q ordered set of concepts for each one of the 20 selected transitions. Apparently, a straightforward solution for the visualization of the space could be plotting the list of concepts encountered by the path. However, many concepts do not have a straightforward interpretation for a human user and this represents an issue. As an example the concept 'Tokyo' is included in many sentic-computing resources; in fact, without an appropriate contextualization, it is almost impossible for a reader to provide a sentimental interpretation of it. Hence, one should address the issue of providing a meaningful representation for the information extracted by Algorithm 9. Two different approaches are proposed.

The first approach defines a-priori a group of eight tags that can be associated to a concept; eventually, a procedure is entitled to link each concept with its tag. As the tags stem from the hourglass model, the goal is to identify the most active components of the emotional space. The second approach, conversely, associates to each concept 4 analogical values, one for each principal dimension of the hourglass model. Figure 5.10 schematizes the two approaches. The block diagram starts from a pair of known concepts. First, the pair is substituted by Q list of concepts, each one composed on N_c concepts (as per subsection 5.2.2.1). Then, according to the first approach, the lists are processed by using a set of tags and finally, 8 scalar values describe the characteristics of the manifold between the two extremes. According to the second approach, each concept in the list is represented using a 4 dimensional representation and finally the manifold is described by means of 4 histograms. Details about the two methodologies are provided in the following.

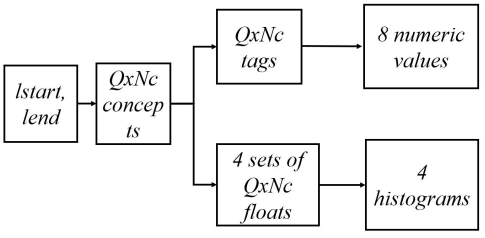


FIGURE 5.10: Block diagram of processing flow

TABLE 5.2: Hourglass model quantization

Name	Positive tag	Negative tag
Aptitude	Admiration	Disgust
Sensitivity	Anger	Fear
Attention	Interest	Surprise
Pleasantness	Joy	Sadness

Tag Descriptors

The first approach tries to model the mechanisms that human brains apply when they perceive emotions: in principle, it is not possible to individuate a specific feeling if it is not reasonably strong. The tag descriptor aims exactly at providing information about the most active sentimental component. From a cognitive view point, one aims to identify the prominent component of the feeling under observation.

Tags are built using the following procedure. Each dimension of the hourglass space is quantized to two values: a positive value and a negative value. Each row of Table 5.2 gives the name of the dimension and the two corresponding tags. Accordingly, for each concept, hourglass activation levels are retrieved; the dimension with the highest absolute value is selected as a descriptor and the sign of the activation provides the final tag. As an example, if a concept has "*Aptitude*" = 0.3, "*Pleasantness*" = -0.4, "*Attention*" = 0.1 and "*Sensitivity*" = -0.7, the leading dimension is *Sensitivity* and the selected tag is "Anger".

For each concept in the transition between two fixed l_{start} and l_{end}, a tag is retrieved, providing a set of Q ordered list of descriptors; for the sake of conciseness, the procedure discard the information related to the order, merge all the list in a single set composed of $N_c \times Q$ tags and consider only the percentage of each tag; all the information related to the order relies in the concepts at the extreme of the path:

$$Per_{tag} = \frac{\#_{tag}}{N_c \times Q} \tag{5.19}$$

where $\#_{tag}$ is the number of times that a specific tag is counted.

The number of repetitions of each tag inside the list is strongly influenced by the total percentage of concepts belonging to the tag cluster. For this reason, in addition to the percentage a second format is presented. In this case, the prior probability of a tag is considered: for the i_{th} tag the prior probability p_{tag} is inferred considering the ratio between the number of concepts available with the i_{th} tag and the total number

TABLE 5.3: Output structure for tag descriptors

Admiration	Disgust	Anger	Fear	Interest	Surprise	Joy	Sadness
Per_{Adm}	Per_{Dis}	Per_{Ang}	Per_{Fear}	Per_{Int}	Per_{Surp}	Per_{Joy}	Per_{Sadn}

of concepts belonging to embedding domain. The prior probability is then applied to renormalize the percentage calculated as per (5.20), with the purpose of weighting more the comparison of less frequent tags and penalize the counter related to most probable ones.

$$Norm\_Per_{tag} = \frac{\alpha \times \#_{tag}}{N_c \times Q \times p_{tag}} \tag{5.20}$$

where α is a normalization factor such that the sum of the percentage of all the tags is 1. This operation is necessary because tag distribution is quite unbalanced in many available resources; this unbalancing introduces a bias in the results that taint results' interpretability.

The final outcome can be reshaped as a column vector with eight columns. Each row represents a tag and tag relative to the same dimension are always adjacent. Table 5.3 contains an explicative example of the transposed version.

The final output consists of four matrices, obtained placing side by side columns relative to subsequent pairs of landmarks. An example can be useful: consider dimension *Sensitivity*; the corresponding output matrix is obtained placing side by side columns relative to: 'rage'→ 'anger', 'anger' → 'annoyance', 'annoyance' → 'apprehension', 'apprehension' → 'fear', 'fear' → 'terror'. This ordered set of columns provides the description of the entire manifold, relative to a dimension of the hourglass model.

Distributions Descriptors

The second set of descriptors is composed of the hourglass representation of each concept. This set is more complete of the simple tag and provides an analogical information about the activation of all the components of the hourglass.

In practice, accordingly to Plutchik's theory this four dimensional representation describes exhaustively the sentiment of a pattern. It is itself an embedding that contains all the sentiment information. The main difference with respect to AffectiveSpace is that AffectiveSpace models explicitly the interactions between different concepts, while Hourglass representation targets only a correct modeling of the affective information of

the single concept. For this reason, this set of descriptor can be useful to illustrate consistently all the affective information embedded in a single point of the affective space i.e. the concept.

Similarly to the case of tag descriptors, the procedure merges all the concepts related to a single transition, obtaining $N_c \times Q$ descriptors. Each of them is represented as a 4 dimensional vector with values between $[-1; 1]$ based on the hourglass' activations. For each feature, the value distribution is computed using a histogram with 6 equally spaced bins (i.e the number of quantization in the hourglass). In the end, for each transition four columns are presented, one for each independent dimension.

As in the case of tag descriptors adjacent columns related to adjacent transitions are placed side by side, providing a set of 4 matrices that describe the numeric distribution of all the values in the manifold.

5.2.3 Experimental Setup

The experimental campain consists in the characterization of AffectiveSpace. This analysis serves not only as a benchmark for the proposed methodology but also to provide a topological analysis of one of the most commonly used concepts embedding. The subsection first provide some details about the construction of this embedding, followed by some implementation details about the cognitive analysis tool.

5.2.3.1 Space Characteristics

AffectiveSpace is a compressed version of AffectNet that is an affective commonsense knowledge base developed upon the graph representation of the Open Mind corpus, called ConceptNet [159], and WordNet-Affect [160], a linguistic resource for the lexical representation of affect. It consists on a semantic network where multi-word expressions of commonsense knowledge are nodes and the links between these are relations between concepts (Figure 5.11). A matrix representation of AffectNet is achieved by dividing each assertion into two parts: the first is the concept and the second is simply the assertion with the first or the second concept left unspecified such as 'is a kind of liquid' or 'a wheel is part of'. Numerical values are obtained considering the reliability of the assertions; practically, positive or negative numbers are associated to single assertions based on the reliability, with a magnitude that grows logarithmically with the confidence score. Finally, to visualize the concept-relation-concept structure of the graph in a matrix format, the data are reshaped with every known concept of some statement being a row and every known semantic feature (relationship + concept) being a column. Matrix

representation involves a series of advantages including the possibility of performing cumulative analogy, executed by first selecting a set of nearest neighbors (in terms of similarity) of the input concept and then by projecting known properties of this set onto unknown properties of the concept.

Even if powerful, the matrix representation of AffectNet consists in thousands of columns involving a series of computational issues that limit its use on many practical applications. To address this problem, Cambria et al. proposed AffectiveSpace [1] that is an embedding for concepts built by means of random projections on the matrix representation of AffectNet with the purpose of compressing the semantic features associated with commonsense concepts and, hence, better performing analogical reasoning on these. It consists of 100,000 concepts in a 100 dimensional space. This high dimensional space, is at the core of the so called Sentic Computing Engine [161]. Even if practically used in many application, little topological information about this model are available; furthermore, all the description available are based only on the pair angle, module of each concept.

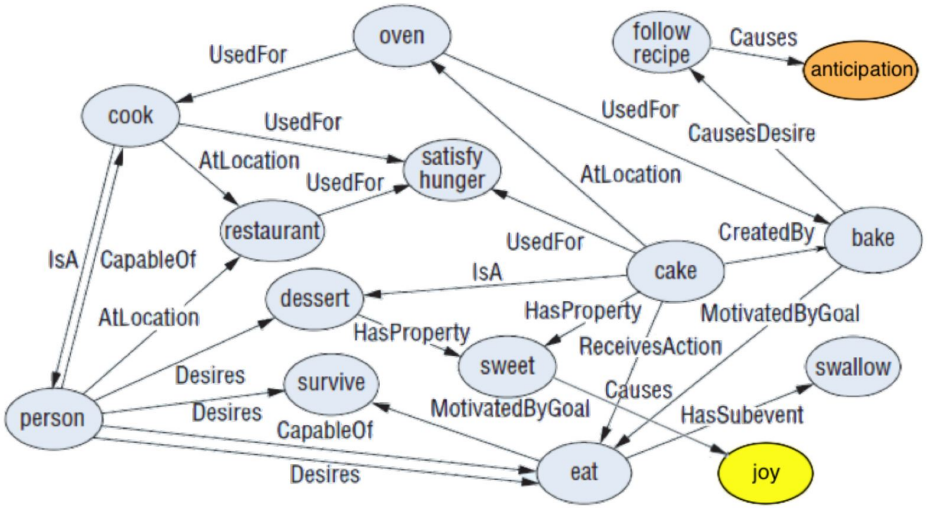


FIGURE 5.11: Example of AffectNet structure for the concept *cake* [2].

Concepts	Semantic Features (*relationship+concept*)							
	..	*Causes* joy	*IsA* event	*UsedFor* housekeeping	*LocatedAt* party_venue	*PartOf* celebration	*MotivatedByGoal* clean_room	..
⋮		⋮	⋮	⋮	⋮	⋮	⋮	
wedding	..	0.94	0.86	0	0.79	0.88	0	..
broom	..	0	0	0.83	0	0	0.87	..
buy_cake	..	**?**	**0.78**	**0**	**0.80**	**0.91**	**0**	..
birthday	..	0.97	0.85	0	0.99	0.98	0	..
sweep_floor	..	0	0	0.79	0	0	0.91	..
⋮		⋮	⋮	⋮	⋮	⋮	⋮	

FIGURE 5.12: Example of AffectNetMatrix structure [2].

In order to characterized this space, a pair of issues should be addressed. As previously said, it consists on 100,000 concepts in a 100 dimensional space; this configuration is by construction sparse. To point out the issue involved by this configuration, consider figure 5.13: this 2 dimensional example show a somehow pathological configuration; the distribution of points is composed of two parts well separated and the limits of the path are set as two points on the extreme sides of the distribution. Most of the prototypes (red crosses) describing the path are outside from data distribution; this phenomenon becomes prominent with the growth of the dimension's number. Should be pointed out that this is not a problem in principle, but it involves some deteriorations in the quality of the solution obtained projecting prototypes on the closest concept.

5.2.3.2 SenticNet

The set of cognitive descriptors is obtained using SenticNet. This resource provides the embedding of concepts in the hourglass's four dimensional space. This resource has the further advantage of containing exactly the same concepts that are mapped in AffectiveSpace, removing the problem of possible mismatches between concepts associated to the prototypes and descriptors available. In cases where the matching is non exact a matching strategy should be developed.

In order to fully exploit this resource, an observation about tags distribution is necessary. Table 5.4 brings the information about the distribution of tags in SenticNet. In the table each raw correspond to a tag paired with the relative percentage respect to the total number of concepts. Table 5.4 enlightens that the probability of different tags is far from being uniform: tags Joy and Sadness correspond to almost 50% of the data, while tags Admiration and Disgust sum up to around 2% of the total. This unbalanced structure of tags distribution is a typical case where prior probability normalization is necessary.

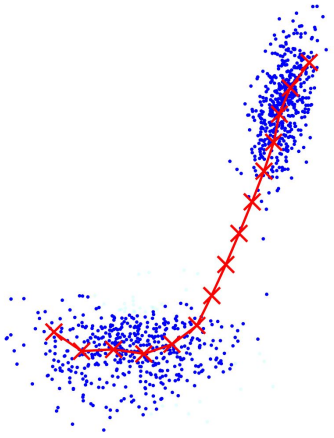


FIGURE 5.13: Example of prototypes in empty regions of the data space

TABLE 5.4: Percentage of tags for the whole dataset

Name	Percentage
admiration	0.0155
disgust	0.0050
anger	0.1237
fear	0.0537
interest	0.1647
surprise	0.0787
joy	0.2893
sadness	0.2696

It is possible to argue that the hourglass representation and tags available in SenticNet are built on top of AffectiveSpace polarizing the analysis, but some important observations should be done: first, numerical values are obtained using a normalized scalar product similarity function, instead in this work results exploits Euclidean distance; second this study analyses the space by means of manifolds, introducing a selection mechanism of prototypes different from the ones used to build the projections in the small dimensional space.

5.2.3.3 Experimental Configuration

The experimental characterization of AffectiveSpace is divided in three phases based on the set of descriptors used. The first part is related to the description of the path using tag descriptors; the second one is related to the projection of concepts in the hourglass space and finally, the last analysis aims to dissolve doubts about polarization of the results due to the projection of the embedding points in concepts.

This experimental design aims to:

- Empirically show the effectiveness of the proposed methodology in visualizing a cognitive description of the space;
- Provide a detailed characterization of AffectiveSpace;

Setup of algorithm parameters are shared by all the different sections of the experimental campaign, in particular, the RKM algorithm has been set in linear configuration, the selected regularization parameters have been obtained using Bayesian Evidence Maximization and the number of clusters is set to 15. The paths are obtained using as metric

the Euclidean distance. In order to perturb the solutions and obtain meaningful information about the manifold structure six sub-sampling of the space have been performed, consisting in the selection of 50,000 concepts (half of the maximum number of concepts).

All the experiments were conducted using Matlab software.

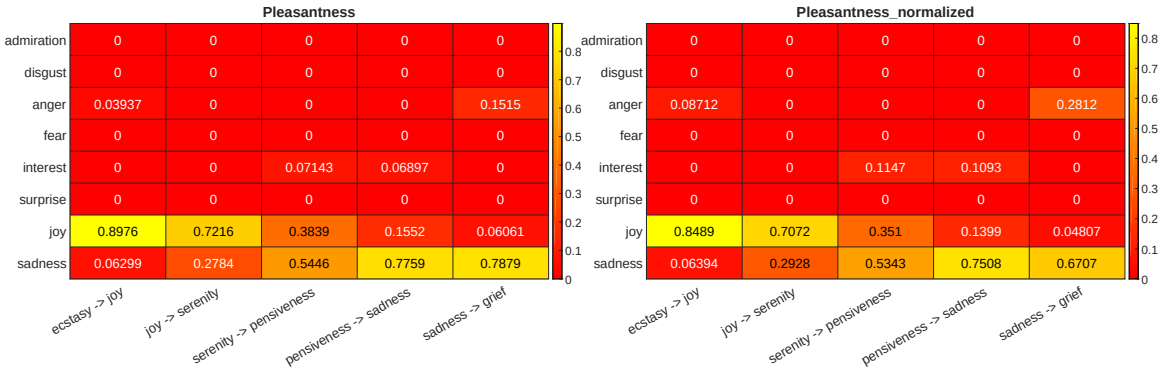
5.2.4 Experimental Results

This subsection divides the results of the experimental campaign in three main parts as introduced in the previous paragraph.

5.2.4.1 Tag Analysis

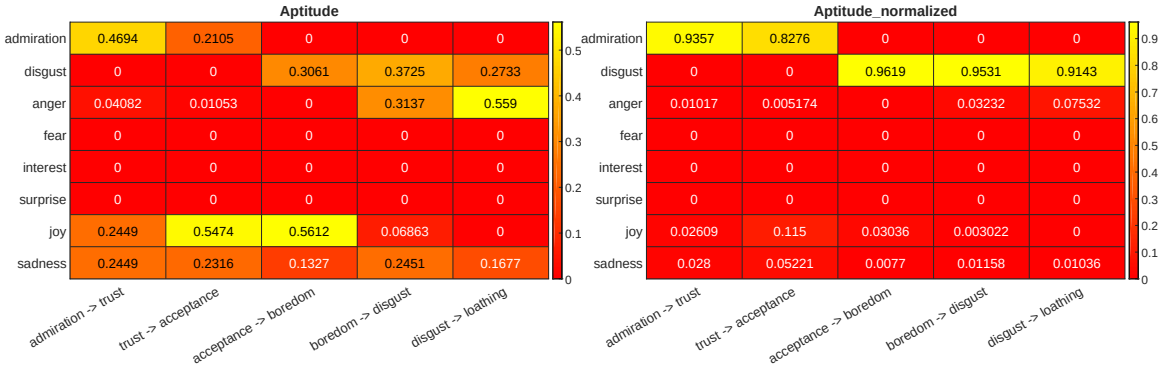
The first visualization proposed involves the descriptors introduced in subsection 5.2.2.2. The space has been characterized by studying the four dimensions of the hourglass model independently, as introduced in subsection 5.2.2.1. Figures 5.14, 5.15, 5.16, 5.17 stand for the four different dimensions of the hourglass space. Each dimension is characterized using the manifold that links the most significant concepts, i.e. the concepts corresponding to the six activation levels (fig. 5.8). All the figures share the same configuration: each figure contains two plots; plot (a) always refers to the case in which the absolute percentage of each tag is considered, as per eq. (5.19), while plot (b) refers to the case in which the percentage of tags is re-normalized respect to the prior probability of each tag (eq. 5.19). All the plots share the same setting: in the x-axis, per each column, the extreme of the morphism under analysis are shown, while the y-axis refers to the tag descriptors, following the structure of table 5.3. To provide a more immediate visualization, percentages of activation are associated with colors. Each cell is painted based on its numerical value, red color means a value of 0% while yellow a value corresponding to the maximum. Should be noted that the color scale is not fixed between different dimension. This configuration has been selected for the sake of readability. In each plot, the corresponding color bar is presented.

There are some evident outcomes: looking at non normalized graphs, one can easily see that, non-surprisingly, *Pleasantness* tag are active in all the 4 transitions, i.e., a considerable percentage of prototypes are associated with that tag. This is a direct consequence of the fact that almost 50% of the concepts are labeled with these 2 values (Joy, Sadness). In addition to these two main labels, a considerable percentage of data are tagged as Anger. In all the cases, except for *Attention*, it is possible to see that tags related to the dimensions under analysis are considerably active, and the activation moves coherently with the path, from positive tag to the negative tag of the dimension



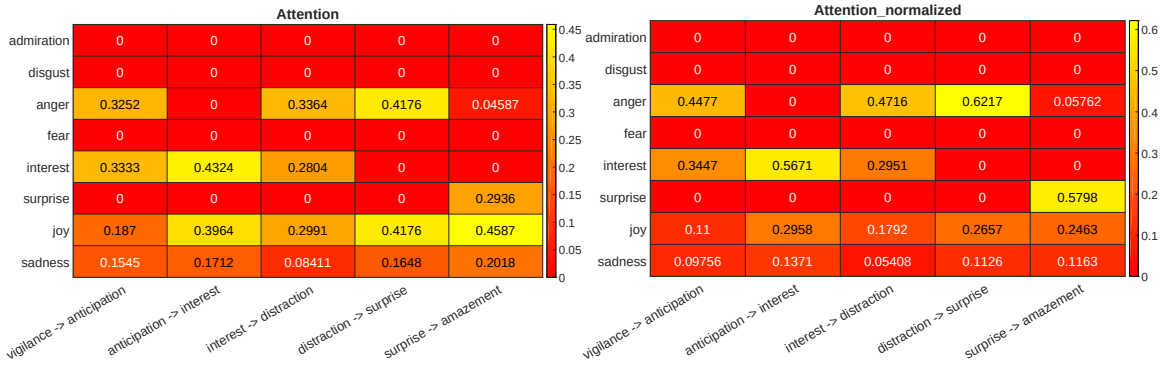
(a) (b)

FIGURE 5.14: *Pleasantness* visualization using tag descriptors.



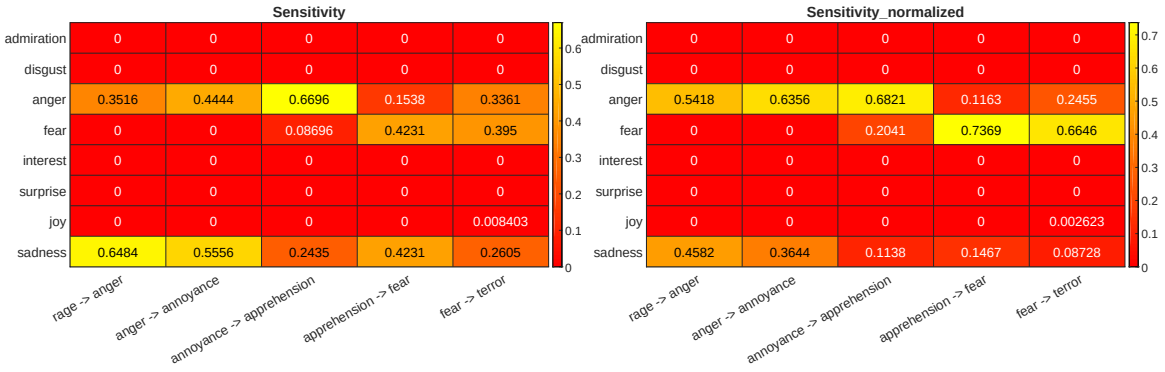
(a) (b)

FIGURE 5.15: Aptitude visualization using tag descriptors.



(a) (b)

FIGURE 5.16: *Attention* visualization using tag descriptors.



(a) (b)

FIGURE 5.17: *Sensitivity* visualization using tag descriptors.

under analysis. Moreover, the re-normalized graphs for transitions of *Pleasantness*, *Aptitude*, *Sensitivity* dimensions show that the weighted distribution of tags involves almost only the tags of the dimension under analysis, coherently with the psychological model. Differently, in transitions related to *Attention* is involved a strong component of anger even after the normalization process.

As a major outcome of the proposed analysis, even if the space is by construction polarized respect to the *Pleasantness* dimension, when re-normalized percentage is considered, one can see how the levels of activation of different concepts show consistency with the cognitive nature of the space.

Finally, the results for each dimension are discussed separately. *Pleasantness* (Fig. 5.14) is the dimension related to the most active tags. The path shown in the graph depicts a clear transition coherent with the investigation direction; some noise appears in the last step, i.e., Sadness to Grief, where 16% of the data are associated with Anger. In any case, from a cognitive point of view, this "mixing" is reasonable. In *Aptitude* dimension (Fig. 5.15) it is evident the noise introduced by the unbalanced nature of the space, especially because Admiration and Disgust tags are associated to only 2% of the concepts; re-normalization with respect to the prior probability unveils that the list of concepts encountered during the transition is coherent with the nature of the tag. Different is the outcome for *Attention* dimension (Fig. 5.16) where Interest tag is coherent with the transition between the concepts but Surprise does not appear until the last transition; even after re-normalization tag related to Joy, Sadness and Anger are quite active. Finally, for *Sensitivity* dimension (Fig. 5.17) holds the same observation done for *Aptitude*.

5.2.4.2 Distributions Analysis

The second analysis proposed involves the descriptors introduced in subsection 5.2.2.2. The outcomes of the analysis of each dimension are composed of four images, organized in a unique figure. All the subplots share the same configuration. The x-axis represents the transitions between concepts associated with the dimension under analysis, following the same setup of the previous experimental campaign. The y-axis refers to the descriptors 5.2.2.2, i.e. the activation levels of the corresponding dimension. Subplot (a) always refers to the distribution of the values relative to dimension *Aptitude*, (b) *Attention*, (c) *Pleasantness* and finally (d) *Sensitivity*. In practice, each column of the matrix is the histogram of the distribution of activation levels. The bins of the histogram correspond to the 6 activation levels individuated by Plutchik's theories. Miming the configuration of the previous set of images, color scale is not common to all the plots.

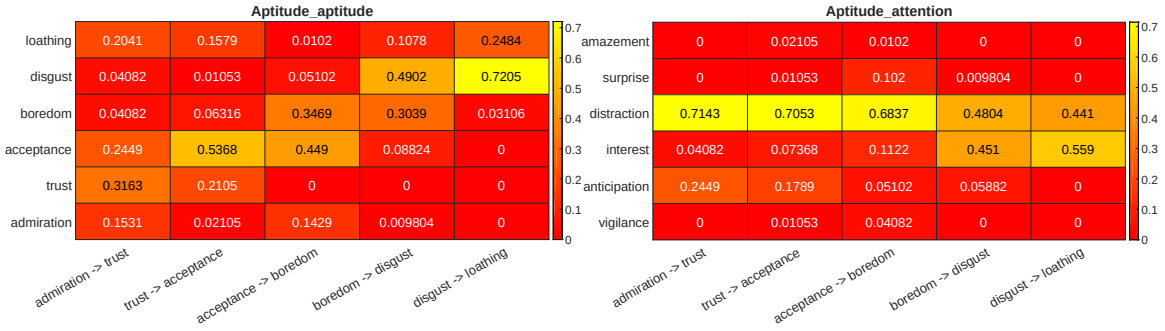
Ideally, if the dimensions would be independent, when studying the transition between the archetypal concepts of one of them, all the histograms would not be affected by such transition, excepted for the one of the dimensions under analysis. In other words, 3 sub-figures should show a constant line, while the one corresponding to the transition under analysis should contain a straight line with angular coefficient equal to one.

Results are reported in figures 5.18, 5.19, 5.20, 5.21. The first and most important observation is that, coherently with psychological theory, follow the transitions inside a single dimension, provides changes in the distribution relative to almost only that dimension. This empirically confirms the independent behavior of these four quantities. It is important to point out that these transitions are dimension-wise because of the paths choice, that should be orthogonal to other sets for the selection of the concepts.

Consistent changes in values distribution along the path can be found only in numerical distributions related to same values, i.e., *Aptitude*/*Aptitude*, *Attention*/*Attention Pleasantness*/*Pleasantness* and *Sensitivity*/*Sensitivity*. For example in Figure 5.18, the only dimension in which we can see a transition of the concept moving thought this direction is *Aptitude* itself (Fig. 5.18(a)). In all the other cases, the other distributions remain almost constant. Similar observations can be done for all the other dimensions. In all the cases a considerable component of noise that does not compromise the qualitative analysis is present.

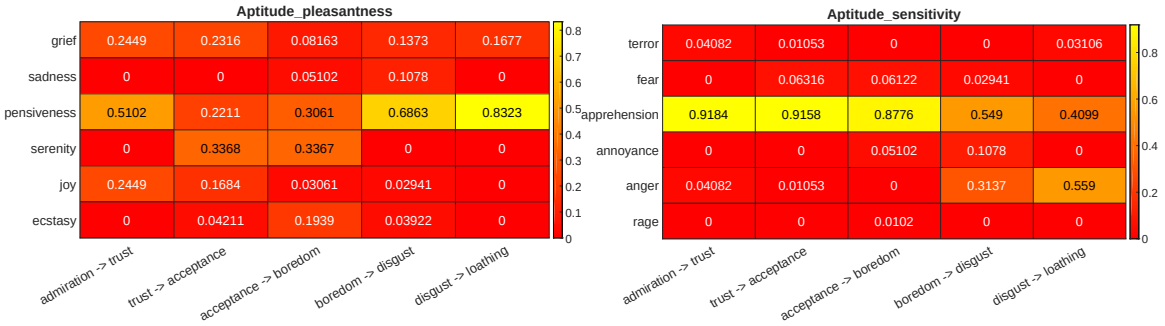
5.2.4.3 Unique Tag

In principle, one could argue that the number of prototypes strongly affects the cognitive description of the space because multiple prototypes can be assigned to the same real



(a) Aptitude

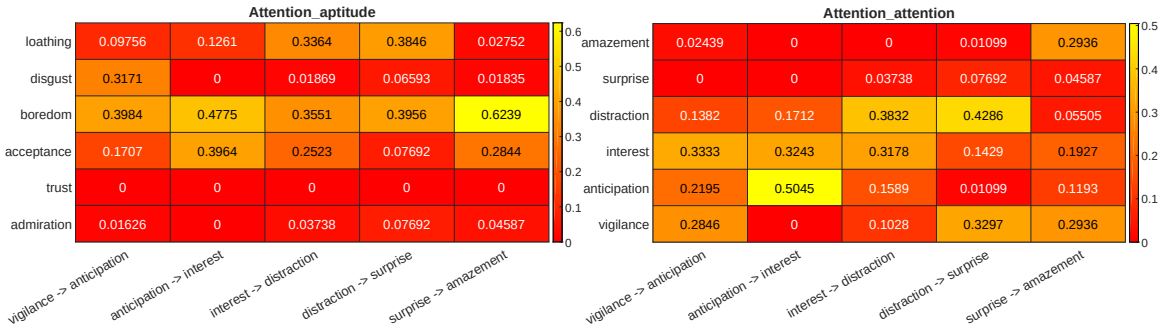
(b) Attention



(c) Pleasantness

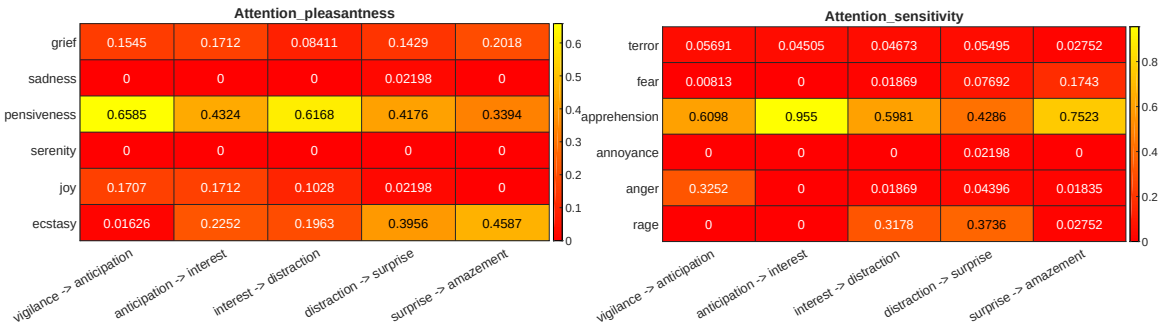
(d) Sensitivity

FIGURE 5.18: *Aptitude* visualization using distributions descriptors.



(a) Aptitude

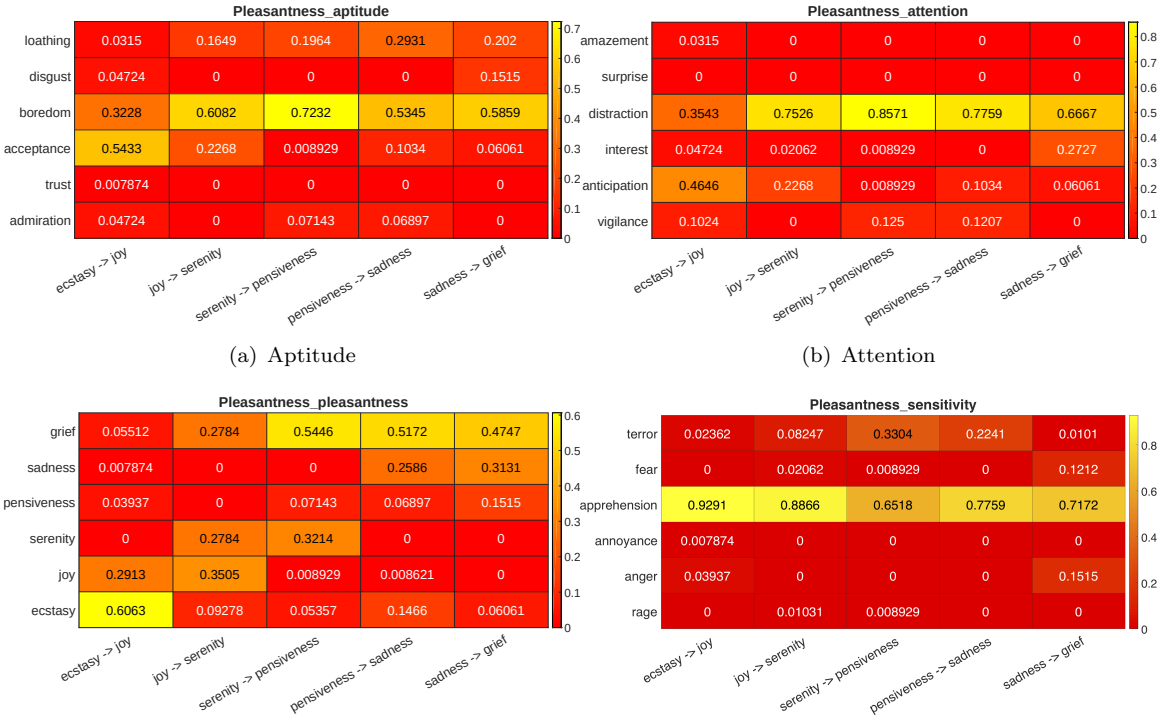
(b) Attention



(c) Pleasantness

(d) Sensitivity

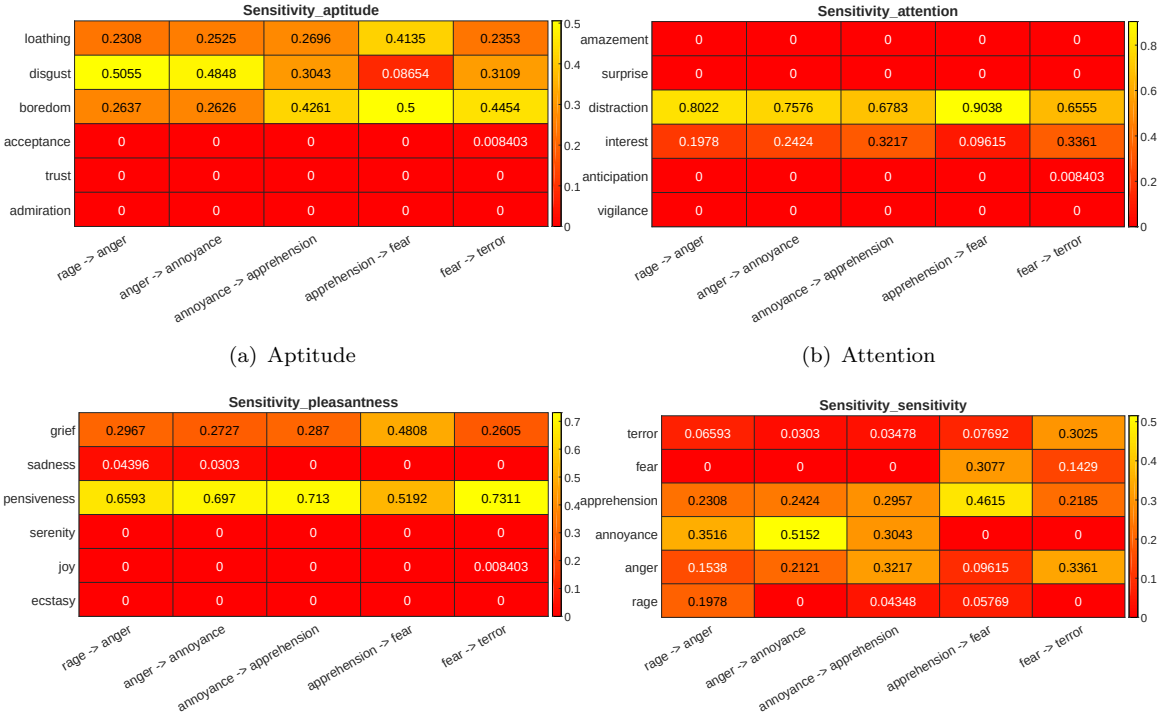
FIGURE 5.19: *Attention* visualization using distributions descriptors.



(a) Aptitude (b) Attention

(c) Pleasantness (d) Sensitivity

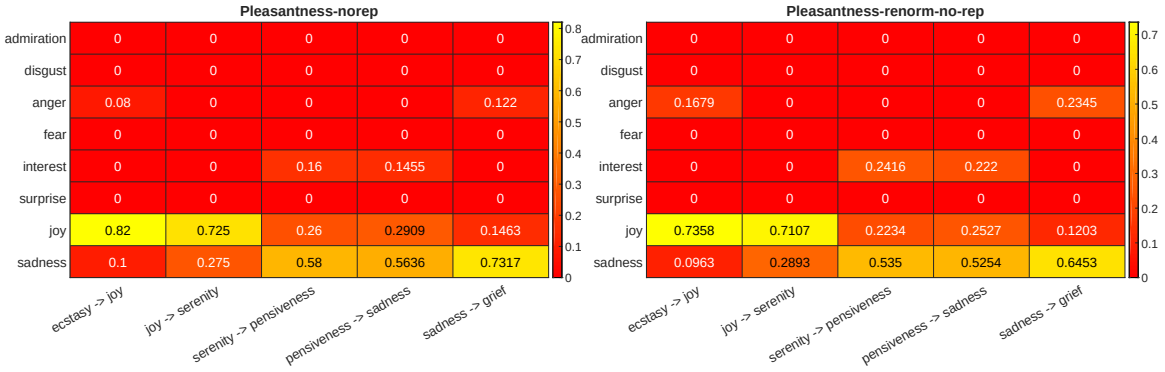
FIGURE 5.20: *Pleasantness* visualization using distributions descriptors.



(a) Aptitude (b) Attention

(c) Pleasantness (d) Sensitivity

FIGURE 5.21: Sensitivity visualization using distributions descriptors.



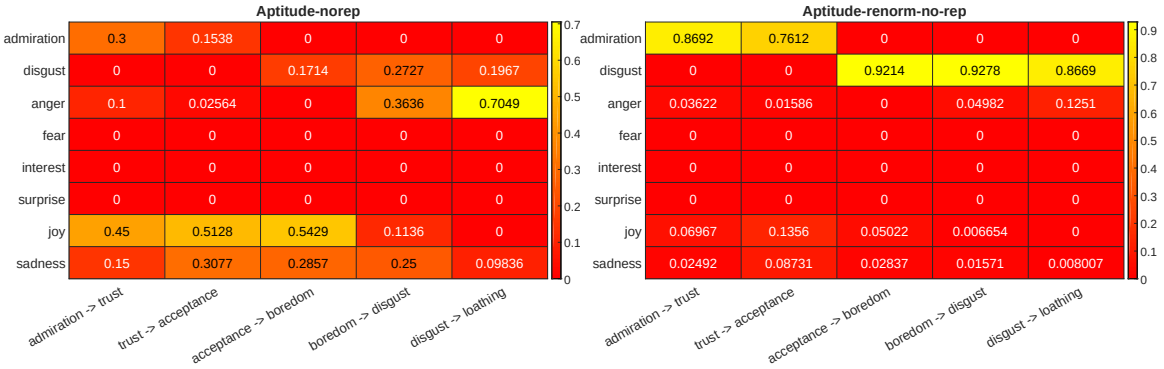
(a) True (b) Re-normalized

FIGURE 5.22: *Pleasantness* visualization using tag descriptors with not repeated concepts.

concept. In fact, with high probability, many prototypes can be assigned to the same concept when using projections of points in the numeric space to exact concepts, given the sparse nature of the space. As a result, it is possible that a single concept in an underpopulated area of the space, can be selected multiple times to represent the prototypes, even if the distance between the true concept and the prototypes is big. This phenomena can introduce a bias in the results. Consider the example depicted in figure 5.13 for an intuitive explanation.

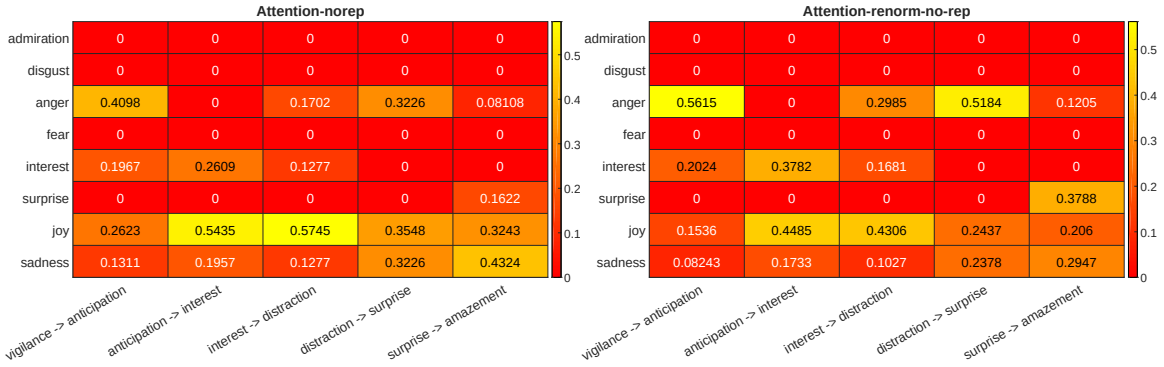
To study this issue, a set of simulations related to the tag descriptors is performed, using the same setup of the first campaign. The only difference is that, in each path, only unique concepts are chosen. To exemplify, consider the extreme case of a path composed of 15 prototypes. Assume that, all the prototypes are projected on the same concept, except for two cases. Following the proposed approach, 13/15 of the information would be associated with a single concept. Instead, in this experimental section, the contributions of the 3 concepts are considered equal. From a cognitive point of view is evident that this approximation is too strong. On the other hand, this configuration deletes possible polarization of the results.

The results are depicted in figures 5.23, 5.24, 5.22, 5.25 using the same configuration presented in previous subsection 5.2.4.1. Looking at the outcome of this set of simulations, fluctuations in the numerical values can be observed but the trends are consistent with the ones shown before in the first part of the experimental campaign, empirically proving that the number of clusters does not substantially changes the results.



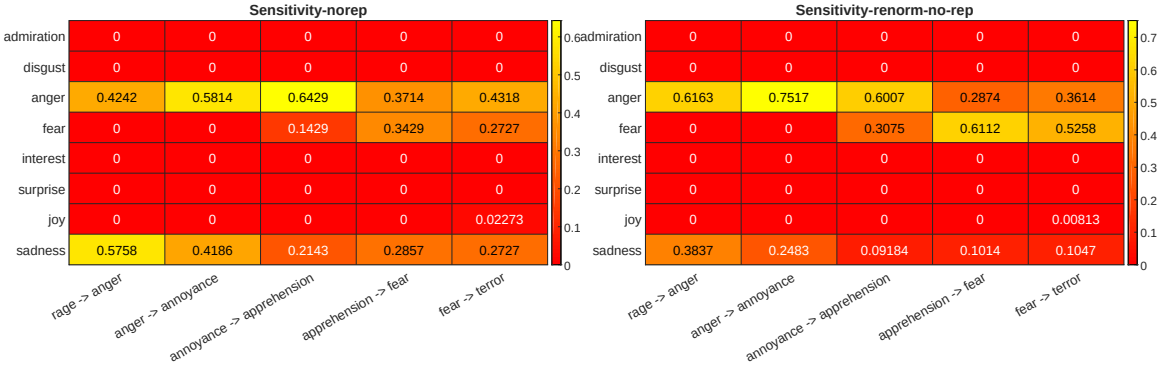
(a) True (b) Re-normalized

FIGURE 5.23: *Aptitude* visualization using tag descriptors with not repeated concepts.



(a) True (b) Re-normalized

FIGURE 5.24: *Attention* visualization using tag descriptors with not repeated concepts.



(a) True (b) Re-normalized

FIGURE 5.25: Sensitivity visualization using tag descriptors with not repeated concepts.

5.2.5 Concluding Remarks

This research presents a methodology based on principal paths in data space and psychologically motivated descriptors to characterize high dimensional spaces induced by embeddings for concepts. The work focused on the properties of AffectiveSpace, providing the visualization of a set of cognitive descriptors that enlightened a series of congruences between concepts distributions in that space and the hourglass of sentiment model.

Chapter 6

Application: Image polarity detection

"A picture is worth a thousand words" is an English idiom that is becoming every day more actual. Mainly thanks to social networks, images have become one of the most important communication tools in the world. Every day users share millions of images and videos; in most of the cases, these contents are expected to convey affective information. As a result, in recent years applications of sentiment analysis started covering multimedia resources [162–164] in addition to text-only resources [165–168].

Sentiment analysis is a branch of data mining which aims to aggregate emotions and feelings from different types of documents. Image polarity detection indeed addresses a specific task within this framework: to distinguish images that raise positive emotions in human users from images that cause bad sentiments [169–172]. Image polarity detection stimulates interest both from industry and academia, as its applications are countless, e.g.: human-robot interaction, stock market prediction, political forecasting and social network analysis. In fact, in recent years, several works addressed this relatively new topic [162, 163, 173].

Actually, deep networks made available a powerful tool to automate the process of feature extraction, which may prove challenging when dealing with complex, 2-dimensional sources of information such as images. Such aspect becomes relevant when considering that polarity detection can be accomplished only by understanding the interaction between different components of an image. In this regard, deep networks can effectively support the task of object recognition, which is a prerequisite to characterizing interaction between objects.

Polarity however cannot be assessed by simply analyzing interactions between objects in an image. Polarity also depends on the context in which the image is examined; similar objects and even similar images can induce different polarities in different scenarios. Figure 6.1 [3] is a shiny example of a picture that can be interpreted in different ways based on the cultural background and historical moment. The rainbow flags have been associated to pacifist movements in the past and nowadays are mainly related to homosexual rights. In general, in the era of social media, content marketing, and custom profiles, this means that one might need to re-train the polarity detection strategy within a short time interval (minutes rather than days) to address the change of circumstances. In fact, re-training a deep network in the context of image sentiment analysis poses major problems. First, the availability of a huge amount of labeled data cannot be taken for granted. Second, the complexity of the learning procedure might hinder fast training, unless massive computational resources are available.

Transfer learning techniques [174] provide an effective solution to both problems. Transfer learning allows one to use a deep network trained on a given domain in the design of a new predictor that tackles a different domain. In some cases, the setup of the eventual predictor might not even require the fine tuning of the pre-trained network, i.e., the update of its parameters by means of a learning procedure involving the new domain. Anyway, the fine tuning process proves faster than training from scratch. The literature indeed shows that image polarity detection can achieve excellent performance by exploiting transfer learning [162, 163, 173]. Nonetheless, a few crucial issues remain open. First the literature lacks of a fair, comprehensive comparison between the various setups that have been proposed to combine deep networks and transfer learning techniques. Second, little attention has given in the past to the trade-off between the computational cost associated to the transfer learning process and the eventual accuracy of the predictor.

This research aims at addressing these open issues by defining a unambiguous design of experiment for the comparison of different approaches to image polarity detection. In this regard, three different configurations have been analyzed, which correspond to core of transfer learning literature for the task of polarity detection. The evaluation process involved the most prominent architectures based on convolutional neural networks (CNNs), which proved outstanding in the area of object recognition. The eventual predictors were mainly compared according to two attributes: classification accuracy and computational cost of the training process. This set of premises leads to an analysis that tests the necessary characteristic for the development of these demanding models on resource constrained devices. As a major consequence, this work provides a strong baseline

FIGURE 6.1: The polarity assigned to this image mostly depends on the cultural background [3].

for the development of systems for the evaluation of image polarity that present a satisfying trade-off between computational cost and accuracy. Five well-known datasets provided the benchmarks for the experimental evaluation.

The rest of the Chapter is organized as follow. Section 6.1 provides a brief overview of the different architectures adopted in the area of object recognition. Section 6.2 analyzes the literature to report on the different designs that supported the implementation of image polarity detection via transfer learning. Section 6.3 defines the design of experiment that has been exploited to fairly evaluate the role played by CNNs in polarity detection frameworks; accordingly, the three different configurations adopted in the design of the predictor are introduced. Sec. 6.4 characterizes the three configurations in terms of computational complexity of the training process. In section 6.5 the outcomes of the experimental sessions involving the benchmarks are showed and discussed. Finally, Section 6.6 provides a few concluding remarks.

6.1 CNNs for Object Recognition

CNNs have been obtaining outstanding results in the task of object detection. This section briefly reviews the most significant CNN architectures, which indeed have been already used in the development of frameworks addressing image polarity detection.

The *AlexNet* [175] architecture represents the first deep network that achieved important results in the field of image classification. In its standard configuration, this architecture included a total of 8 weight layers (5 convolutional layers and 3 fully connected layers); overall, *AlexNet* was characterized by $61 \cdot 10^6$ parameters. Such aspect obviously posed major problems in terms of computational load, as classification performance was strictly

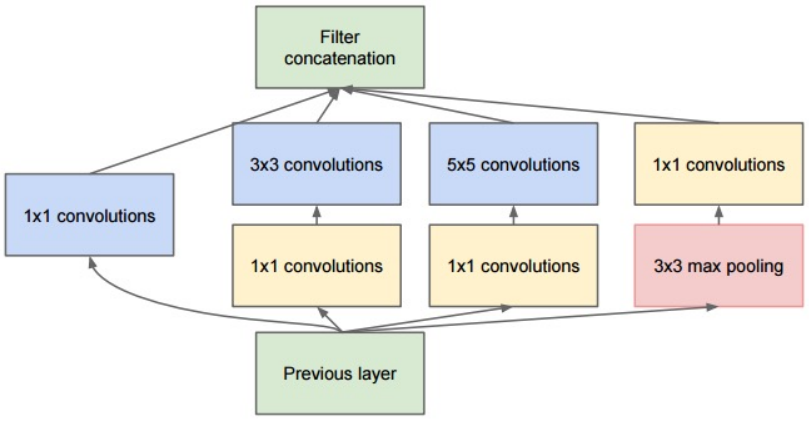


FIGURE 6.2: Example of inception module: the output of "Previous layer" feeds 4 different feature extractors. The informations provided from the different branches are merged by the layer "Filter concatenation";

related to the availability of very large training sets. This issue was suitably tackled by exploiting the computational capabilities of Graphic Processing Units (GPUs), which provided a fundamental tool to deal with the trade-off between training time and size of the training set. Such setup actually opened the way to the massive use of GPUs as computational units for deep learning.

Vgg_16 and *Vgg_19* [176] improved the architecture of *AlexNet* by increasing the number of weight layers to 16 and 19, respectively; besides, these architecture exploited exclusively stacks of 3×3 convolutions. Both *Vgg_16* and *Vgg_19* proved able to outperform AlexNet in terms of classification performance. On the other hand, they also inflated the number of parameters to be set by the learning process.

In 2014, *GoogLeNet* [177] introduced major novelties in the design of CNN architectures by proposing the *Inception Module*. Such module actually was entitled to implement a local, small network topology; as a result, *GoogLeNet* relied on a stack of Inception Modules. An example of the module is shown in Fig. 6.2. First, the output of the previous layer feeds different filter operations, which are completed in parallel. Then, the outcome of the different filters is merged into a single third-order tensor by projecting the resulting depth to a lower dimension. Such organization allowed the eventual stack of Inception Modules to limit the amount of parameters involved. In its standard configuration, *GoogLeNet* featured 22 weight layers and still involved less parameters than *AlexNet*. Indeed, this architecture has been updated several time (e.g., *Inc_V3* [178] and *Inc_v4* [179]).

ResNet architectures [180] addressed the design of very deep networks. These architectures exploited as building blocks *Residual networks*, which replaced standard convolutional blocks; the underlying goal was to deal with the convergence issues brought about

by deep networks. To introduce the *Residual block*, it is convenient to denote as $\mathbf{H}(\mathbf{x})$ the mapping to be fit by a few stacked convolutional layers, where $\mathbf{x}$ is the input to the first layer; accordingly, $\mathbf{F}(\mathbf{x}) := \mathbf{H}(\mathbf{x}) - \mathbf{x}$ is the residual function approximated by such stacked layers (under the assumption that $\mathbf{H}(\mathbf{x})$ and $\mathbf{x}$ have the same dimensionality). Figure 6.3 schematizes the general structure of a Residual block, which is entitled to eventually yield $\mathbf{F}(\mathbf{x}) + \mathbf{x}$. Overall, the architecture was organized as a stack of such blocks, in which each block was trained on the residual representation of the previous one. Thus, *Res_50*, *Res_101*, and *Res_152* increased the number of weight layers to 50, 101, and 152, respectively.

Table 6.1 provides the main details of the architectures discussed above. For each network the table provides: the top-1 accuracy achieved on ImageNet [181] competition (in percentage); the number of weight layers; the total amount of parameters; the number of floating point operations performed for a single image classification [182]. The amount of parameters involved basically affects two aspects: memory occupation and expected size of the training set. On the one hand, when starting with a pre-trained network, transfer learning can reduce the amount of patterns required to successfully complete fine tuning. On the other hand, the size of such training set is still strictly related to the number of parameters. The number of floating point operations provides a suitable approximation of the overall computational complexity that characterizes an architecture. In fact, a detailed metric would take into account that, in general, different instructions have different computational costs (e.g., memory accesses might severely affect computational performance). The table shows that *GoogLeNet* is the most convenient choice in terms of both parameters and computational complexity. However, the other architectures (except *AlexNet*) proved able to outperform *GoogLeNet* in terms of accuracy.

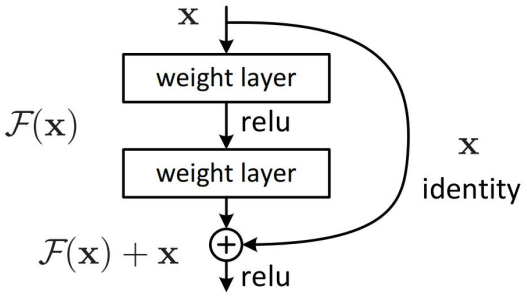


FIGURE 6.3: Shortcut module: graphical representation of residual concept.

6.2 Image Sentiment Analysis: State of the Art

Image polarity detection is an emerging topic in the area of sentiment analysis. Indeed, this topic has been covered by a few interesting surveys [162, 163, 173] recently. However, these previous works do not focus specifically on the role played by CNN architectures in the development and ongoing improvement of image polarity detection frameworks. In this regard, it is worth to mention that several important works addressed image polarity detection before that deep learning changed the approach to image processing [183–191]. However, currently models based on CNNs represent the state of the art in this area.

In the following, Subsection 6.2.1 will report on the most interesting approaches to polarity detection based on CNNs. Subsection 6.2.2, on the other hand, will briefly survey models that did not target specifically polarity detection but indeed included an image sentiment analysis module. Finally, Subsection. 6.2.3 analyzes the most significant benchmarks in the area of image polarity detection.

6.2.1 Polarity Detection

In general, polarity detection is tackled by inheriting the core structure from object recognition frameworks. Thus, according to the transfer learning setting, one deals with subjectivity detection by exploiting the low-level features extracted by a CNN trained on object recognition. In practice, such approach leads to two main different designs, as schematized in Figure 6.4. In the first design (Fig. 6.4(a)), the trained CNN is subject to changes only in the layers entitled to implement classification. Usually, the last fully connected layer is replaced with a new fully connected layer that includes as many neurons as the classes involved in the polarity detection problem. An alternative option

TABLE 6.1: Attributes of different CNN architectures exploited in the area of object recognition

Architecture	Layers	Top-1 Acc (%)	Operations (Gflops)	Parameters ($\cdot 10^6$)
AlexNet	8	54	<u>0.7</u>	61
Vgg_16	16	71	15.5	138
Vgg_19	19	71	19.6	144
GoogLeNet	58	68	1.6	<u>7</u>
Inc_v3	46	78	6	24
Res_50	50	76	3.9	26
Res_101	101	77	7.6	45
Res_152	152	<u>79</u>	11.3	60

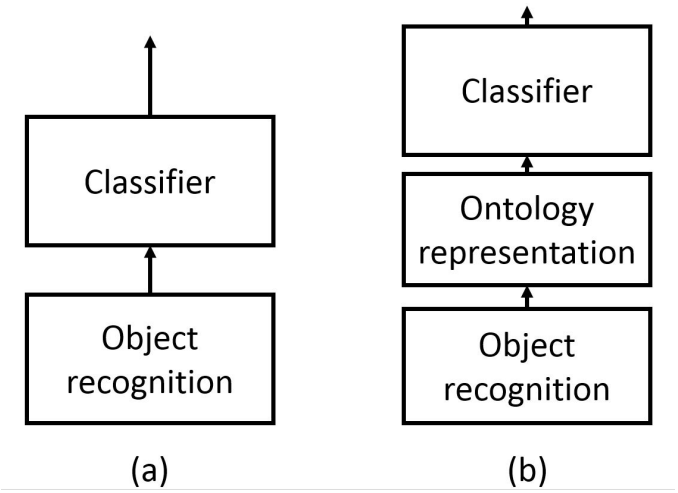


FIGURE 6.4: Designs of polarity detection frameworks. (a) The low-level features provided by the pre-trained CNN feeds a classifier entitled to tackle polarity detection. (b) The low-level features provided by the pre-trained CNN feeds a module entitled to model an ontology; such mid-level representation feeds the classifier entitled to tackle polarity detection.

is to feed a given classifier with the low-level features provided by the CNN. By taking advantage of fine tuning, one can also re-train the CNN to tailor its parameters on the specific polarity detection problem. In the second design (Fig. 6.4(b)) a two-step procedure leads to the eventual classifier. First, the trained CNN is augmented by replacing the last fully connected layer with a new architecture, which should model an ontology. The eventual layout is re-trained as a whole by using a Visual Sentiment Ontology (VSO). In the literature, several works relied on adjective noun pairs (ANPs) [169, 170] to complete this task: these ontologies provide a collection of images for a given pair {adjective, noun}. In the second step, such new layout becomes the building block of the polarity detector. Hence, one may replace the last fully connected layer, analogously to the first design. A second option is to add a given classifier on top of the layout; in this case, one uses as input of the classifier the probability distribution of the classes that characterized the underlying ontology. A new training process then is applied to the final layout. It is worth noting that usually the first step relies on weak labels in the learning process, as VSOs are generated by automated algorithms that add noise to the labeling action.

The literature provides different actual implementations of the two designs. Principally, the implementations differ in the choice of the CNN to be adopted, the transfer learning technique, and the training data domain. In the same year two distinct works exploited *AlexNet* as a low-level feature extractor in the design of (Fig. 6.4(a)). In [169] a logistic regression layer provided the classifier; the authors compared two different solutions: 1) the classifier replaced the last fully connected layer in the *AlexNet* architecture; 2) the classifier was stacked over the the *AlexNet* architecture. In [192] a support vector

machine (SVM) was used as classifier. Campos et. al [193] indeed exploited the *AlexNet* architecture as feature extractor to complete an interesting analysis on the effects of layer ablation and layer addition in the eventual accuracy of the polarity detection framework. A design involving ontology representation (Fig. 6.4(b)) was implemented in [194]; the authors proposed a layout for sentiment ontologies recognition based on the *AlexNet* architecture, in which the last fully connected layer had one neuron for each ANP. This approach outperformed the previous SentiBank classifier based on SVM [189]. Jou et al. [195] further improved the predictor presented in [194] by exploiting a multilingual visual sentiment ontology (MVSO). In [170] the authors proposed a custom architecture for image polarity prediction: the model stacked two convolutional layers and four fully connected layers. The training procedure involved weak labels provided by ANP. In fact, the authors applied a peculiar multi-step learning process: after each step, the dataset was pruned by removing the images correctly classified. An empirical comparison between such architecture, the architectures analyzed in [193], and the architecture proposed in [195] was presented in [172]. In this work, the authors indeed evaluated the role played by the specific ontology in the transfer learning process. The best performance was obtained by models relying on the *AlexNet* architecture and the English version of MVSO. The *Vgg_19* architecture and the *GoogLeNet* architecture were exploited, respectively, in [171, 196]. Interestingly, both papers showed that such architectures proved able to support predictors that improved over state-of-the-art predictors based on the *AlexNet* architecture.

The enhancement of ontology representation in the design of (Fig. 6.4(b)) has been the specific goal of a few works. Fernandez et al. [197] proposed a framework in which two independent CNNs are adopted to learn ANPs: one network was designed to deal with nouns, the other network was designed to deal with adjectives. Both predictors exploited the *Res_50* architecture. A similar approach was utilized in [198]; the eventual framework however relied on a different strategy when merging the information provided by the two CNNs. Wang et al. [199] indeed proposed a model where adjectives and noun are predicted by different CNNs; nonetheless, the two networks are coupled by a mutual supervision mechanism. A modified version of *Res_50* supported the model proposed in [200]; in that case, the residual learning mechanism was extended to multi-task cross-residual learning. As a result, a single network was able to explicitly consider the interactions between different ANP couples (e.g.: 'shiny-cars', 'shiny-shoes').

CNN architectures also supported models for image polarity detection that does not fit into either of the two designs described above. In [196], the *Vgg_19* architecture processes an image including the conventional R, G, and B and a focal channel; the additional channel is set to model human attention. Similarly, [201] studied the role played by saliency in sentiment detection. Attention mechanisms [202, 203] have been

also explored in [204], where the authors utilized *VggNet* architectures in combination with a recurrent neural network (RNN). The RNN allows the model to explore an image as a composition of small areas, thus mimicking the human attention process where a person focuses only on the most important parts of the image.

6.2.2 Sentiment Analysis: Other Applications

The automated generation of image captions with sentiment terms is indeed a task closely related to sentiment prediction: the underlying goal is still to provide an efficient method for extracting affective information from images. Actually, the true difference resides in how information is conveyed, as the final user for a caption generation model is a human. The archetypal model [205] in this area was composed by a CNN architecture that fed a Gated RNNs [206]. Here, the CNN was entitled to extract low-level features from the image, while a long short-term memory (LSTM) network modeled the textual description of the image. In [207, 208] a *VggNet* architecture has been adopted to extract low-level features; in [209], such task was addressed by a *Res_152* architecture. Karayil et al. [210], conversely, exploited the *AlexNet* architecture to model ANPs; a multi-directed graph eventually ranked the ANP couples provided by the CNN, thus generating captions. To the purpose of improving sentiment description, Sun et al. [211] extended a pre-trained caption generation model with an emotion classifier to add abstract knowledge.

Multimodal sentiment analysis also can take advantage of reliable models for image polarity detection. Remarkable results have been achieved in [3, 212–215] where ensembles of handcrafted features were extracted from images and then combined with information provided by text analysis. However, such approaches have been outperformed by frameworks that utilize CNN for extracting features from visual contents [170, 215–218].

In [199, 219, 220] image sentiment analysis has been modeled as a probability distribution problem. The common rationale behind the three different approaches is that a single image can evoke different feelings at the same time. From a methodological view point, though, these works did not introduce any novelty about feature extraction via CNN.

It is worth mentioning, finally, that polarity and sentiment detection is also exploited in face analysis [221–225], posture analysis [226] and the analysis of groups of people [227]. These methodologies are indeed tailored for specific tasks, and strictly related to video domain.

TABLE 6.2: Benchmarks for image sentiment prediction

Name	# data	# classes	add info
Twitter1 [170]	882	2	n.a.
Twitter2 [189]	603	2	text
Twitter3 [3]	19600	3	text
Flickr [213]	105.587	2	text
Instagram [213]	120.000	2	text
IAPS [228]	394	8	n.a.
Art photo [185]	807	8	n.a.
Abstract paintings[185]	228	8	n.a.
MVSA[229]	23.308	8	n.a.
Twitter_LDL [230]	10,045	8	n.a.
Flickr_LDL [230]	10,700	8	n.a.
Visual Realism [196]	2520	2	n.a
ANP [189]	1M	3244	adjective name pairs
MVSO(EN) [195]	4M	4422	adjective name pairs

6.2.3 Benchmarks

Table 6.2 lists the most common benchmarks in the area of image sentiment prediction; the table purposely does not include datasets designed for caption generation, faces sentiment detection and multi modal analysis. In Table 6.2, the first column gives the name of the dataset along with the reference paper; the second column gives the size of the dataset; the third column gives the number of classes; finally, the fourth column details the additional information provided with the images, if any.

All the benchmarks involving 2 classes explicitly refer to polarity detection. The dataset Twitter3 can be considered as a special case, as it also includes the class *neutral*. Except for Twitter1, these benchmarks also provide the textual message linked to an image. The benchmarks involving 8 classes actually refer to emotion recognition. The last two datasets of Table 6.2 provide two examples of ontologies, where a collection of images is associated to a given adjective noun pair. Such datasets can support the development of the design showed in Fig. 6.4(b).

6.3 A Compared Analysis

The literature proves that CNNs are widely utilized as feature extractor (in the transfer learning setup) in all the frameworks that need to rely on image sentiment analysis.

Indeed, in general, these frameworks differ both in terms of specific design of the feature extraction process and of overall design of the polarity predictor; thus, one deals with heterogeneous setups that may be different for the training dataset or for the use of fine tuning strategies. As a major consequence, it is never easy to evaluate the actual contribution provided by a given architecture in boosting the performance of image polarity detection.

The present work wants to address this issue by proposing a formal experimental protocol that should support a fair comparison between different configurations of the feature extraction process. To this purpose, the design shown in Fig. 6.4(a) is used as reference. Such design is indeed implemented according to three distinct configurations, as showed in Fig. 6.5. These configurations differ in the specific structure of the classification block that processes the output of the pre-trained CNN and in learning strategy applied to the resulting predictor, which may either involve or not the adoption of a fine tuning procedure for the parameters of the pre-trained architecture. In practice, the proposed configurations covers the most important approaches presented in the literature. In the figure, bold characters identify blocks that are trained from scratch in the learning process of the polarity detector, while italic characters identify pre-trained blocks that are subject to fine tuning. The details of the three configurations will be discussed in subsections 6.3.1, 6.3.2, and, 6.3.3, respectively.

In principle, the configurations schematized in Fig. 6.5 allows one to suitably evaluate the role played by the specific architecture in the predictor. In this regard, it is worth noting that this work aims at analyzing mainly two aspects: the accuracy of the predictor and the computational load involved in the corresponding learning procedure. Actually, configurations that might stem from the design of Fig. 6.4(b) would not be as useful in supporting a fair comparison between different architectures. In fact, the intermediate layer set to model ontologies could actually introduce biases brought about both by the configuration of this layer and by the adopted ANP; the latter element is indeed critical as ontologies might also add a cultural bias [195]. In the proposed configurations, an unbiased background has been set by relying only on CNNs pre-trained using the ImageNet dataset [181], a well-established benchmark for the object recognition area.

6.3.1 Layer Replacement

The configuration schematized in Fig. 6.5 (a) refers to the most common outline for image polarity detection. Here, the last fully connected layer of a CNN trained on object recognition is replaced by a new fully connected layer, which serves as classification layer for the eventual predictor and includes as many neurons as the number of classes

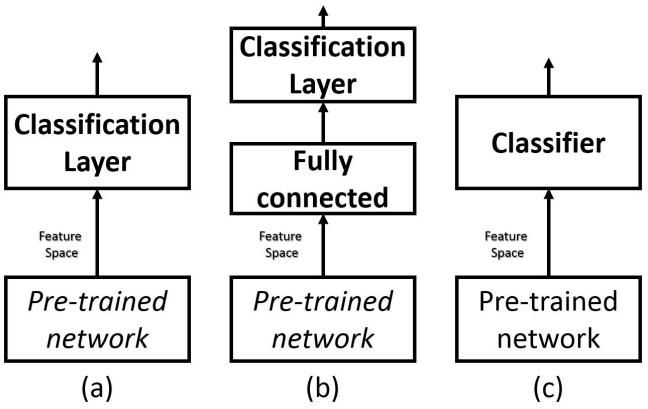


FIGURE 6.5: Three different configurations of the polarity predictor: (a) Layer Replacement; (b) Layer Addition; (c) Classifier. Blocks with bold characters refer to modules that are retrained from scratch; blocks with italic characters refers to modules that are subject to fine tuning

represented in the polarity problem. Accordingly, the eventual predictor is a universal approximator combining a feature extraction block that inherits its parameterization from object recognition and a classification layer that should be trained from scratch. In the learning procedure, the predictor is trained on the new domain by utilizing a small number of epochs. Training is indeed extended to the feature extraction block, thus adopting fine tuning; by suitably setting the learning rate, the involved parameters are only subject to small perturbations [174].

Actually, similar, yet alternative configurations might be obtained by exploiting different strategies of layer ablation. Campos et al. [172], though, has showed that replacing the last fully connected layer represents the most convenient choice when addressing image sentiment analysis. This outcome seems reasonable when considering that the last fully connected layer is entitled to model the probability distribution of different objects in the image.

6.3.2 Layer Addition

The second configuration (Fig. 6.5 (b)) augments the standard layout presented above. The building block is still provided by a pre-trained CNN that had the last fully connected layer removed. In this case, though, this block feeds a fully connected layer including N_h neurons; the classification layer is then stacked on top of such intermediate layer. Therefore, in the resulting layout one has a Single Layer Feed-forward Network (SLFN) that processes the features provided by the CNN. As SLFNs are universal approximators themselves, this configuration does not need to rely on the linear separability of data in the feature space. This element differentiates such configuration from the first configuration, in which a linear separator processes the data represented

in the feature space. Accordingly, in the Layer Replacement configuration fine tuning is expected to suitably adjust the feature space.

The learning procedure resembles the one adopted for the first configuration. Thus, training involves a small number of epochs. The main goal is to train from scratch the SLFN, while the parameters of the feature extraction block are subject to fine tuning. In principle, the training process is made easier and thus faster by the presence of the SLFN, as convergence might be reached without satisfying linear separability in the feature space. On the other hand, one should deal with two possible drawbacks: 1) N_h represents an hyper-parameter that should be properly set; 2) this configuration is prone to overfitting, as one expects to utilize relatively small training sets.

6.3.3 Classifier

The third configuration (Fig. 6.5 (c)) inherits from the second configuration the overall concept: using a universal approximator to classify the data that lies in the feature space provided by the CNN. The eventual implementation indeed utilizes a SVM as classifier. In general, other classifiers could play the same role in this configuration; however, SVM combines excellent generalization performance with a convex optimization problem.

In the proposed layout, the pre-trained CNN (without the last fully connected layer) feeds a SVM that exploits a Gaussian kernel. The learning procedure is actually entitled to train the SVM classifier by using standard convex optimization methods, which remove any issue brought about by the presence of local minima. In fact, such setup precludes any fine tuning process for the feature extraction block, as back-propagation mechanisms are not involved. This in turn means that one has to fully rely on the feature space tuned on the object recognition domain. Such aspect may represent a drawback that stems from the choice of adopting a convex optimization problem. Nonetheless, one should take into account that the Gaussian kernel, as almost any kernel, involves hyper-parameters.

6.4 Computational Complexity

The three configurations presented above can be firstly analyzed according to the computational load involved in the learning process of the eventual predictor. Such analysis does not need to rely on empirical protocols, as the related computational costs can be formally assessed.

In the case of the first configuration, Layer Replacement, the computational cost of the training procedure O_{lr} can be approximated by defining the following quantities:

- n_{tg}: number of training samples;
- n_{ep}: number of epochs;
- n_{lay}: number of layers;
- $n_{par,i}$: number of parameters in the ith layer;
- O_{ff}: computational cost of a feed-forward step;
- O_{bw}: computational cost of a backward step, which involves gradient computation and parameters'update.

The choice of the architecture directly impacts on the last four quantities. Accordingly, O_{lr} can be expressed as

$$\begin{aligned} O_{lr} = n_{ep} * n_{tg} * [(O_{ff}(P) + O_{bw}(P)] = \\ n_{ep} * n_{tg} * [(O_{ff}(P) + \alpha * O_{ff}(P)] = \\ n_{ep} * n_{tg} * (\alpha + 1)[(O_{ff}(P)]; \end{aligned} \tag{6.1}$$

where

$$P = \sum_{i=1}^{n_{lay}} n_{par,i}^3. \tag{6.2}$$

Empirical evidence suggests that the value of the coefficient α in (6.1) can be roughly approximated with 2 [182].

Equation (6.1) and (6.2) show that both O_{ff} and O_{bw} scale cubically with the number of parameters. Indeed, (6.2) indicates that an architecture is not only characterized by the total number of parameters. In the case of uneven distribution of the parameters, the computational cost is mostly determined by the largest layers (in terms of weights).

Equation (6.1), in practice, also approximates the computational cost O_{la} of the training procedure for the second configuration, Layer Addition. Actually, such configuration only add a single fully connected layer to the layout of the first configuration; besides, this layer is expected to include a small number of neurons ($N_h < 500$), as small or medium size datasets are involved. Thus, $O_{la} \approx O_{lr}$.

The third configuration, conversely, relies on a different learning process, as fine tuning of the pre-trained CNN is not requested. Hence, the training procedure basically includes two steps: 1) to project all the training samples into the feature space defined by the CNN, and 2) to train the SVM on those data by exploiting a convex optimization problem. Let O_{SVM} be the computational cost of SVM training; then, the computational cost O_{cl} of the overall training procedure can be expressed as:

$$O_{cl} = O_{ff}(P) * n_{tg} + O_{SVM}(n_{tg}^3). \tag{6.3}$$

In Equation (6.3), O_{ff} scales cubically with the number of parameters in the largest layer of the architecture, while O_{SVM} scales cubically with the number of training samples. Therefore, the second component is expected to prevail over the first component only when large training sets are involved, as $n_{tg} > n_{par,i}$ for any i.

A comparison between the three configurations cannot be completed without taking into account a few elements that do not emerge in equations (6.1) and (6.3). First, all the configurations need -in principle- to rely on model selection; this in turn means that the actual learning process encompasses multiple runs of the training algorithm. In this regard, the second configuration might be more computationally demanding than the first configuration, even if $O_{la} \approx O_{lr}$, as one adds N_h to the parameters to be set by model selection.

Second, the issue of sub-optimal solutions affects heuristic optimization techniques such as stochastic gradient descent. As transfer learning relies on small datasets, such issue may plausibly perturb the training of polarity predictors based on the first and second configurations. Besides, the risk of being trapped in local minima is increased by the adoption of early stopping strategies, which on the other side prevent overfitting. Actually, multi-start optimization represents a suitable solution. However, this solution also inflates the computational load of the learning procedure, which might require multiple runs of the training algorithm. On the contrary, the third configuration does not suffer from the issue of local minima, as it fully relies on a convex optimization problem. As a major consequence, such configuration may be preferred when one wants to avoid the computationally demanding multi-start optimization.

Third, the actual execution time of a CNN architecture might not depend only on P. On the one hand, GPUs are designed to fully exploit data and model parallelism. On the other hand, CNNs are realized as a stack of layers that should be completed sequentially. As a result, the execution time of architectures such as *Vgg_16* and *Vgg_19* might be lower than the execution time of *Res_152*. While the latter architecture is undeniably lighter in terms of parameters, both *Vgg_16* and *Vgg_19* involve a sensibly smaller

number of layers. Accordingly, GPUs supported by large memories can suitably take advantage of the configuration {small number of layers, several parameters per layer} to boost computational time.

6.5 Experimental Results

The experimental campaign aimed at assessing the generalization ability of the deep convolutional neural networks inherited by object recognition. The CNNs are tested based on the configurations presented in Section 6.3. Indeed, the goal was also to evaluate how the design of the feature-extraction module influenced the performance of a given configuration. For the sake of consistency, all the experiments were implemented in MATLAB®with the Neural Network Toolbox, which provided the pre-trained versions of the architectures listed in Table 6.1. Actually, *Res_152* was the only architecture not included in the experiments, as the Neural Network Toolbox did not provide its implementation.

6.5.1 Datasets

The experiments involved five datasets: Twitter1 (Tw_1) [170], Twitter2 (Tw_2) [189], Twitter3 (Tw_3a and Tw_3b) [3], and ANP40 [189]. Table 6.3 gives respectively, for each dataset, the total number of patterns, the number of patterns belonging to the class "positive polarity", and the number of patterns belonging to the class "negative polarity".

Twitter1 is the most common benchmark for image polarity recognition. In its original version, the dataset collected 1269 images obtained from image tweets, i.e., Twitter messages that also contain an image. All images have been labeled via an Amazon Mechanical Turk (AMT) experiment. This experimental session actually utilized the "5 agree" version of the dataset. Such version includes only the images for which all the five human assessors agreed on the same label. Thus, the eventual dataset included a total of 882 images (581 labeled as "positive" and 301 labeled as "negative").

Twitter2 [189] is a second dataset that collects image tweets; it includes a total of 603 images. Labeling was indeed completed via an AMT experiment. Eventually, 470 images were labeled as "positive" and 133 images were labeled as "negative". Such dataset indeed represents a challenging benchmark as 1) it is small, and 2) it is very unbalanced.

TABLE 6.3: Benchmarks adopted in the experimental sessions

Name	# data	# positive	# negative
Tw_1 [170]	882	581	301
Tw_2 [189]	603	470	133
Tw_3a [3]	702	351	351
Tw_3b [3]	702	351	351
ANP40 [189]	17114	11857	5257

Twitter3 also provides a collection of image tweets [3]. In this case, tweets were filtered according to a predetermined vocabulary of 406 emotional words. Thus, the dataset incorporated only the tweets that contained in the text message (hashtags included) at least one of those words. The vocabulary encompassed ten distinct categories covering most of the feelings of human beings (e.g., happiness and depression). Three annotators labeled the pairs {text,image} by using a three-value scale: positive, negative, and neutral; text and image were annotated separately. This work utilized a pruned version of the dataset: only images for which all the annotators agreed on the same label were employed, thus obtaining a total of 4109 images. As only 351 images out of 4109 were labeled as negative, such pruned version of Twitter3 resulted very unbalanced. Thus, two different balanced datasets (Tw_3a and Tw_3b) were generated by adding to the 351 "negative" images two different subsets of 351 images randomly extracted from the total amount of "positive" images.

The ANP dataset implements an ontology and is composed of a set of Flickr images [189]. The dataset includes 3,316 adjective noun pairs; for each pair, at most 1,000 images were provided, thus leading to a total of about one million images. The ANP tags assigned from Flickr users have been utilized as labels for the images. Thus, noise affects severely this dataset. To tackle this issue, a pruned version of the dataset is proposed. Accordingly, the 20 ANPs with the highest polarity values and the 20 ANPs with the lowest polarity values were selected (the website of the dataset' authors [1] was exploited as reference). Eventually, the dataset involved in the experiments included 11857 "positive" images and 5257 "negative" images.

6.5.2 Experimental Setup

The experimental campaign encompassed two different sessions. In the first session, Tw_1, Tw_3a, Tw_3b, and ANP40 were used as benchmarks. The first three benchmarks were utilized to assess the performance of the three configurations presented in

[1] http://visual-sentiment-ontology.appspot.com/

Sec. 6.3 under the most common scenario: a small training set is available. On the other hand, ANP40 covered the case in which a large dataset is available. The second session involved only Tw_2 as benchmark. Such session was separated from the first session in consequence of the peculiar properties of this dataset, which is small and also very unbalanced.

In all the session, the setup of the Layer Replacement configuration was organized as follows. Stochastic Gradient Descent with Momentum was exploited as optimization strategy, with momentum = 0.9 and learning rate = 10^{-4} for all the layer belonging to the pre-trained CNN. In the classification layer the learning rate was set to 10^{-3} and the regularization parameter was set to 0.5. Early stopping was adopted, with validation patience set to 2. Training involved a maximum of 10 epochs.

The same setup was adopted for the Layer Addition configuration. In this case, the learning rate was set to 10^{-3} and the regularization parameter was set to 0.5 both for the fully-connected layer and the classification layer. Besides, cross validation was utilized to set the hyper-parameter N_h, which admitted the following values: {10, 50, 100, 200}.

In the Classifier configuration fine-tuning is not involved. The SVM with Gaussian kernel was trained by using a standard algorithm. The two hyper-parameters, regularization term C and kernel standard deviation σ, were set via model selection.

6.5.3 Experimental Session #1

According to the proposed experimental design, seven different pre-trained architectures were utilized to generate as many implementations of each single configuration. The experimental session aimed at assessing and comparing the generalization performance of such implementations.

Given a configuration, the performance of the seven predictors has been evaluated by exploiting a 5-fold strategy. Hence, for each dataset, the classification accuracy of a predictor has been measured via five different experiments, corresponding to as many different training/test pairs. Indeed, five separate runs of each single experiment were completed; each run involved a different composition of the mini-batch. As a result, given a predictor and a benchmark, 25 measurements of the classification accuracy on the test set were eventually available.

Table 6.4 reports on the performance obtained with the Layer Replacement configuration on Tw_1, Tw_3a, Tw_3b, and ANP40. In each row, the first column indicates the pre-trained architecture adopted in the implementation of the predictor; the second column

gives -for the Tw_1 dataset- the average accuracy obtained by the predictor over the 25 experiments, along with its standard uncertainty (between brackets); the third, fourth, and fifth columns give the same quantities for the experiments involving Tw_3a, Tw_3b, and ANP40, respectively. Experimental outcomes showed that *Res_101* was able to slightly outperform the other architectures on Tw_1 and Tw_3a. *Vgg_19* scored the best average accuracy on Tw_3b; the gap between such predictor and both *Vgg_16* and *Inc_v3*, though, was very small, especially if one considers the corresponding standard uncertainties. Analogously, one can conclude that *Inc_v3* and *Res_101* shared the role of best predictor on ANP40. Indeed, it is worth noting that six predictors out of seven achieved an average accuracy ranging between 79.7% and 78.7%; in practice, only the predictor based on *AlexNet* proved to be slightly less effective on ANP40. Such balance possibly stems from the availability of a larger dataset.

The experiments involving the Layer Replacement configuration can also reveal which architecture has been the most consistent over the different dataset and the different training/set pairs. Thus, for each benchmark and for each training/set pair, the seven architectures have been ranked according to the classification accuracy scored by the corresponding predictor. In this case, the classification accuracy associated to a predictor was the best accuracy over the five runs completed for a given training/set pair. In each rank, one point was assigned to the best predictor, two points were assigned to the second best predictor, and so on until the worst predictor, which took seven points. Accordingly, Figure 6.6 provides, for each architecture, the total points marked over the 20 ranks (5 training/test pairs $\times$ 4 benchmarks). Thus, in principle, a predictor could not mark less than 20 points, which meant first position in the rank (i.e., best predictor) for every rank. The plot shows that the predictors based on *Vgg_16* and *Vgg_19* proved to be the most consistent, i.e., these predictors often occupied the highest positions in the rank over the different experiments. Such outcome is indeed consistent with the results reported in Table 6.4, which showed that *Vgg_16* and *Vgg_19* were always able to score effective performance over the different benchmarks.

Table 6.5 reports on the performance obtained with the Layer Addition configuration on Tw_1, Tw_3a, Tw_3b, and ANP40. The table follows the same format of Table 6.4. In each of these experiments, the training set was further split into the actual training set and a validation set to support the model-selection procedure that led to the setup of N_h in the eventual predictor. Experimental outcomes show that *Res_101* was able to score the best average accuracy on Tw_1, Tw_3a, Tw_3b. Indeed, the gap with the second best predictor was always small, especially in the case of the experiments involving Tw_3b. On ANP40, again, six predictors out of seven almost achieved the same average accuracy. This in turn confirmed that the availability of a large dataset somewhat counterbalanced the differences that might exist between the different architectures. Overall,

TABLE 6.4: Experimental Results: Session #1, Layer Replacement

Arch.	Tw_1	Tw_3a	Tw_3b	ANP40
AlexNet	82.5 (0.6)	66.2 (0.5)	65.4 (0.7)	76.3 (0.3)
Vgg_16	86.6 (0.4)	68.9 (1.0)	70.7 (1.0)	79.0 (0.3)
Vgg_19	86.4 (0.5)	69.2 (0.7)	71.0 (0.8)	78.7 (0.3)
GoogLeNet	84.2 (0.5)	66.0 (0.6)	68.2 (0.5)	79.2 (0.3)
Inc_v3	86.5 (0.7)	68.1 (0.7)	70.5 (0.9)	79.9 (0.2)
Res_50	85.8 (0.6)	68.9 (0.9)	68.8 (1.5)	79.2 (0.2)
Res_101	88.2 (0.4)	70.8 (0.6)	69.2 (0.8)	79.7 (0.2)

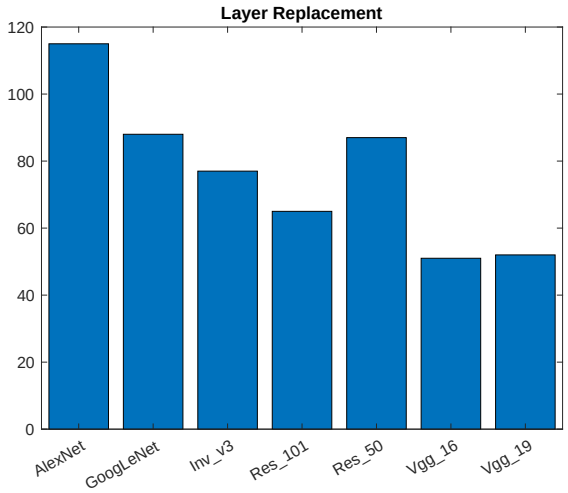


FIGURE 6.6: Consistency evaluation of the seven architectures under the Layer Replacement configuration; the plot gives the architectures on the x-axis and the corresponding total amount of collected points on the y-axis

it is interesting to emphasize that under the Layer Addition configuration the predictors attained better average accuracies than under the Layer Replacement configuration. Such improvement comes at the cost of a more complex training procedure, as the Layer Addition configuration involves model selection for the setup of N_h.

Figure 6.7 provides the outcomes of the consistency assessment for the Layer Addition configuration. As above, the plot gives -for each architecture- the total points marked over the 20 ranks (5 training/test pairs $\times$ 4 benchmarks). In this case, again, *Vgg_19* proved to be the most consistent architecture; however, it did not share its status with *Vgg_16*, which collected as much points as *Res_101*.

Table 6.6 reports on the performance obtained with the Classification configuration on Tw_1, Tw_3a, Tw_3b. The table again follows the same format of Table 6.4. In this case, though, ANP40 was not included in the experiments, as (6.3) suggested that such configuration should be avoided for computational reasons when large training sets are

TABLE 6.5: Experimental Results: Session #1, Layer Addition

Arch.	Tw_1	Tw_3a	Tw_3b	ANP40
AlexNet	84.9 (0.4)	68.2 (0.4)	69.9 (0.7)	78.6 (0.2)
Vgg_16	88.0 (0.3)	70.9 (0.6)	72.3 (0.8)	80.0 (0.3)
Vgg_19	87.6 (0.4)	71.0 (0.7)	73.4 (0.6)	80.3 (0.2)
GoogLeNet	86.3 (0.3)	70.0 (0.4)	71.3 (0.5)	80.0 (0.3)
Inc_v3	85.3 (0.4)	66.4 (0.4)	68.5 (0.7)	80.3 (0.2)
Res_50	87.0 (0.3)	69.4 (0.6)	72.1 (0.8)	80.1 (0.2)
Res_101	89.1 (0.3)	72.1 (0.7)	73.9 (0.6)	80.2 (0.2)

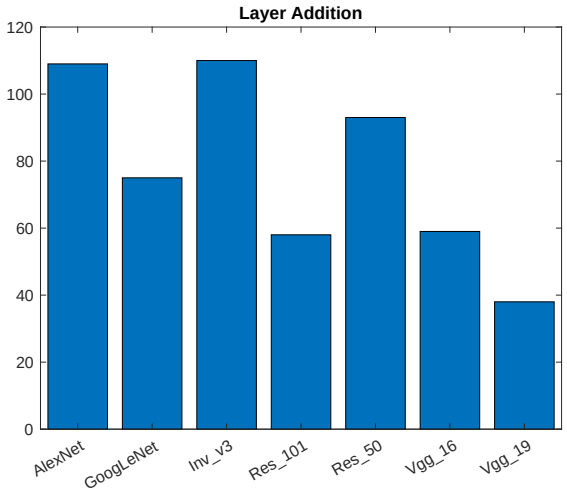


FIGURE 6.7: Consistency evaluation of the seven architectures under the Layer Addition configuration; the plot gives the architectures on the x-axis and the corresponding total amount of collected points on the y-axis

involved. The Classification configuration required a model-selection procedure, similarly the Layer Addition configuration. Hence, in each experiment, the training set was further split into the actual training set and a validation set to support the model-selection procedure that led to the setup of C and σ in the eventual predictor. It is worth noting that predictors based on the Classification configuration relied on a convex optimization problem in the training phase. Thus, the classification accuracy obtained with a given training/test pair could be estimated without resorting to multiple runs of the experiment. As a result, the table gives the average classification accuracy over the five training/test pairs, along with the corresponding standard uncertainty. Experimental outcomes point out the peculiarities of such configuration, which did not exploit fine tuning. First, the predictor based on *Res_101* never scored the best average accuracy. In fact, each benchmark led to a different result in terms of best predictor. Second, the standard uncertainty associated to the average accuracy was always large or even very large. Therefore, given a benchmark and an architecture, the variance of the accuracies

TABLE 6.6: Experimental Results: Session #1, Classifier

Arch.	Tw_1	Tw3_1	Tw_3_2
AlexNet	80.7(3.7)	63.5(3.7)	70.2(1.0)
Vgg_16	86.3(0.1)	70.7(1.0)	70.4(2.3)
Vgg_19	85.3(1.3)	70.4(1.2)	73.3(1.6)
GoogLeNet	79.1(5.3)	67.7(1.3)	69.5(1.7)
Inc_v3	85.1(2.1)	72.1(0.9)	75.2(1.7)
Res_50	84.5(4.6)	72.4(2.3)	72.9(2.4)
Res_101	80.0(5.7)	70.5(1.6)	74.3(1.8)

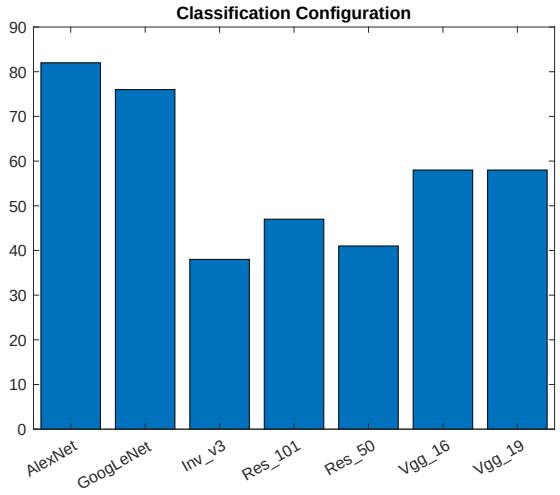


FIGURE 6.8: Consistency evaluation of the seven architectures under the Classifier configuration; the plot gives the architectures on the x-axis and the corresponding total amount of collected points on the y-axis

over the five experiments was always quite wide; that is, the composition of the training set played a major role. In practice, this means that by adopting the Classification configuration one increases the risk of incurring in a weak predictor, independently of the architecture.

Figure 6.8 provides the outcomes of the consistency assessment for the Classification configuration by using the same format adopted above. The plot gives -for each architecture- the total points marked over the 15 ranks (5 training/test pairs $\times$ 3 benchmarks). As expected, this plot is different from those examined above. Under the Classification configuration, *Res_50* and *Inc_v3* proved to be the most consistent architectures, while *Vgg_16* and *Vgg_19* did not attain satisfactory performance.

6.5.4 Experimental Session #2

The dataset Tw_2 poses major challenges to polarity predictors. In fact, a naive classifier that always predicts "positive" would achieve a 78% accuracy on this dataset. On the other hand, the literature proved that is very difficult to attain a predictor that can score satisfactory accuracies on both the classes of this benchmark.

The experimental session revealed that nor the Layer Replacement configuration not the Classification configuration were able to achieve acceptable result on Tw_2. Hence, this section will provide only the outcomes of the experiments involving the Layer Addition configuration. The performance of the corresponding seven predictors has been evaluated by exploiting a 5-fold strategy. Accordingly, the classification accuracy of a predictor has been measured via five different experiments, corresponding to as many different training/test pairs. In each experiment, the originally imbalanced training set was eventually processed to obtain a more balanced set. To this purpose, oversampling [231] was applied to the negative class, i.e., the minority class.

Given a predictor, five separate runs of each single experiment were completed; each run involved a different composition of the mini-batch. Indeed, such routine was repeated over the four admissible values of N_h. As a result, 100 measurements of the classification accuracy on the test set were eventually available for each predictor. Table 6.7 reports on the outcomes of this experimental session by providing -for each predictor (first column)- two quantities: the second column gives the number of successful trials, i.e., the trials that led to a classification accuracy greater than 50% on both the classes; the third column gives the average accuracy over the number of successful trials. The table stimulates two remarks. First, only *Inc_v3* and *Res_50* collected an acceptable amount of successful trails. Second, *Vgg_16*, *Vgg_19*, and *Res_101* proved able to overcome the other architectures in terms of average accuracy in the few cases where successful trials were completed. Thus, on the one hand the latter architectures confirmed to be effective in dealing with polarity detection; on the other hand, they proved to be more prone to problem of local minima.

6.6 Concluding Remarks

This analysis evaluated the role played by CNNs and fine tuning strategies in image polarity detection. The experimental protocol focused on three different configurations of the predictor, which indeed covered the most significant approaches proposed in the literature. The predictors have been analyzed and compared both in terms of generalization performance and in terms of computational complexity. As image polarity

TABLE 6.7: Experimental Results: Session #2, Layer Addition

Arch.	Successful Trials	Accuracy
AlexNet	4	66.5
Vgg_16	10	72.4
Vgg_19	5	72.6
GoogLeNet	8	67.0
Inc_v3	50	62.5
Res_50	45	69.7
Res_101	20	72.9

detection usually relies on small datasets, the assessment of generalization performance allows one to investigate on the ability of a given architecture/configuration to deal with such scenario. At the same time, computational costs may represent a key factor when one should target the implementation of the whole prediction system on an electronic device.

The outcomes of the experimental protocol provided useful insights both on the predictor's configuration (and the underlying fine tuning strategy) and on the convolutional architectures. Among the tested configurations, Layer Addition proved to be the most effective in terms of classification accuracy. This approach is also the most computationally demanding among the three configurations; the training process involves the back propagation procedure and also the model selection procedure that should support the setup N_h. Besides, in principle, the regularization parameter represents indeed an hyper parameter. In this work such parameter was always set to 0.5 to the purpose of exploiting regularization properties of early stopping to tune the fitting degree. On the other hand, by adopting a proper model selection one might be able to improve the generalization performance of a predictor. This obviously implies an increase of the overall computational load.

The Classifier configuration in fact represents the most convenient option from the computational point of view. Unfortunately, this approach proved to be too sensitive to the actual composition of the training set. Such issue makes the application of this approach in real word scenario inadvisable. Conversely, the Layer Replacement configuration achieved a satisfactory balance between accuracy, convergence of the training process, and computational cost. It is worth noting that the training procedure of both Layer Replacement and Layer Addition can be simplified by inhibiting fine tuning in the lowest layers of the CNN. Thus, only the upper layers of the architecture would be involved in the training process. The experimental protocol did not explore this setup

because the networks differ significantly in the number of layers (from 8 to 101) and a fair comparison would be difficult.

Vgg_16, *Vgg_19*, and *Res_101* proved to be the most powerful architectures. In terms of generalization abilities they seemed almost comparable. Indeed, computational aspects may represent a discriminative factor. *VggNets* require the storing of a number of parameters that is almost 3 or 4 time bigger than *Res_101*; furthermore, the forward phase of *VggNets* involves 2 or 3 times the number of floating point operation. However, most of the parameters and operations are introduced in the last fully connected layers of *VggNets*. Thus, such layers may take advantage of parallel computing, which can boost the overall execution time. In *Res_101*, conversely, parameters and operations are evenly distributed among the numerous layers. As a major consequence, parallel computing cannot really improve the execution time. Hence, *Vgg_16* and *Vgg_19* may represent the best option when a GPU with a considerable amount of memory is available. *Res_101* should be preferred when one has to deal with memory constraints.

Chapter 7

Conclusion

This thesis has addressed both theoretical problems and engineering applications of fast learning models. From the theoretical point of view this Thesis mainly contributed with two analysis. Firstly, this dissertation showed that the theory of learning with similarity functions can provide the basis for the development of novel insights on the ELM model. The crucial outcome is that it is possible to reinterpret the ELM mapping layer by introducing the concepts of similarity function and landmark. As a major result, the learning scheme applied by ELM can be described as a viable strategy to search for a consistent (ϵ, γ)-good similarity function. Within this context, the thesis suggested three possible enhancement strategies by focusing on the peculiar role played by the free parameters of the activation/similarity function. These strategies covered different scenarios based on peculiarities of the function at hand. As a result, the contribution of the Thesis consists of ad-hoc training strategies that inherit the advantages of methodologies based on random feed forward neural networks while minimizing the well known waste of computational resources in the inference phase.

A second outcome consists on the investigation of how the theory of learning with similarity functions can support the development of an efficient framework that deals with multi-way data inherently. The notion of similarity is built on the decomposition of a tensor into two components: the core tensor and the corresponding orthonormal basis. The degree of similarity between a pattern and a landmark is then assessed by taking into due account the alignment between the respective basis. This, in turn, means that similarity is not just estimated by conventionally comparing the standard r-mode eigenvalues of the two tensors; this procedure in fact can only provide partial information on the degree of resemblance between the tensors. Nonetheless, it is worth to note that the framework proposed in this thesis can utilize MLSVD to characterize the intrinsic structural properties of each single landmark; as a major result, one can more

reliably capture the local structure of multimodal data manifold. Notably, the proposed solution proved convenient both in term of generalization capabilities and computational cost with respect to state-of-the-art methodologies.

From an applicative point of view, this thesis addressed the efficient implementation of random basis neural network classifiers on low-resources digital reconfigurable devices. The research focused on predictors that exploit the hard-limiter activation function as non linearity. In addition, random networks properties have been exploited to reduce memory requirements. As a result, the Thesis provides two architectures explicitly for implementation on low-power low-cost devices. The two proposal differ in the area/latency trade-off. In particular, the first solution achieves state-of-the-art performances in term of area consumption when implemented on digital devices, meanwhile the second one is characterized by a latency that is almost independent of the input dimensionality.

Lastly, few engineering applications and algorithmic improvements for sentiment detection were discussed and developed in Chapters 5 and 6. Chapter 5 discusses two applications related to sentiment analysis on text. The first outcome is a novel architecture suitable to filter out neutral content in a time and resource effective manner. The framework for subjectivity detection consists in the effective integration of CNNs and the newly proposed BNELM, where the former learns significant features from the training set and, hence, feeds them to the latter. Secondly, the chapter presents an inspection algorithm designed to link psychological models with the properties of numerical embedding for text. This analytic instrument sets the basis for the development of consistent embedding of affective properties of the terms . As a consequence, the inference problem becomes simpler implying a training process and resulting predictors that can be accomplished using relatively simple models.

Finally, Chapter 6 produces a formal evaluation of the role played by deep convolutional architectures in the task of image polarity detection. In particular, existing state-of-the-art architectures are compared in term of generalization performances and computational cost providing a strong starting point for the development of predictors in resource constrained systems.

There are several extensions of the proposed work that could be explored. The implementation on FPGA of the training phase of fast learning approaches described in Chapter 2 would offer a low power solution for the training of classifiers in embedded devices. Similar observations hold for the tensor based approach discussed in Chapter 3. A further extension of the given results involves a deeper study on the application of the techniques proposed in chapters 2, 3 on the applications discussed in chapters 6, 5.

Bibliography

[1] Erik Cambria, Jie Fu, Federica Bisio, and Soujanya Poria. Affectivespace 2: Enabling affective intuition for concept-level sentiment analysis. In *AAAI*, pages 508–514, 2015.

[2] Erik Cambria, Soujanya Poria, Rajiv Bajpai, and Björn Schuller. SenticNet 4: A semantic resource for sentiment analysis based on conceptual primitives. In *COLING*, pages 2666–2677, 2016.

[3] Teng Niu, Shiai Zhu, Lei Pang, and Abdulmotaleb El Saddik. Sentiment analysis on multi-view social data. In *International Conference on Multimedia Modeling*, pages 15–27. Springer, 2016.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Federica Bisio, Paolo Gastaldo, Rodolfo Zunino, Christian Gianoglio, and Edoardo Ragusa. Learning with similarity functions: A novel design for the extreme learning machine. In *Proceedings of ELM-2015 Volume 1*, pages 265–277. Springer, 2016.

[6] Paolo Gastaldo, Federica Bisio, Christian Gianoglio, Edoardo Ragusa, and Rodolfo Zunino. Learning with similarity functions: a novel design for the extreme learning machine. *Neurocomputing*, 261:37–49, 2017.

[7] Edoardo Ragusa, Paolo Gastaldo, Rodolfo Zunino, and Erik Cambria. Balancing computational complexity and generalization ability: a novel design for elm. In *Proceedings of ELM-2018*, page In press. Springer, 2018.

[8] Edoardo Ragusa, Paolo Gastaldo, Rodolfo Zunino, and Erik Cambria. Learning with similarity functions: a tensor-based framework. *Cognitive Computation*, pages 1–19, 2018.

[9] Edoardo Ragusa, Christian Gianoglio, Paolo Gastaldo, and Rodolfo Zunino. A digital implementation of extreme learning machines for resource-constrained devices. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2018.

[10] Christian Gianoglio, Francesco Guastavino, Edoardo Ragusa, Andrea Bruzzone, and Eugenia Torello. Hardware friendly neural network for the pd classification. In *2018 IEEE Conference on Electrical Insulation and Dielectric Phenomena (CEIDP)*, pages 538–541. IEEE, 2018.

[11] Iti Chaturvedi, Edoardo Ragusa, Paolo Gastaldo, Rodolfo Zunino, and Erik Cambria. Bayesian network based extreme learning machine for subjectivity detection. *Journal of The Franklin Institute*, 355(4):1780–1797, 2018.

[12] Vladimir Naumovich Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[13] Gao Huang, Guang-Bin Huang, Shiji Song, and Keyou You. Trends in extreme learning machines: A review. *Neural Networks*, 61:32–48, 2015.

[14] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521 (7553):436, 2015.

[15] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: from big data to big impact. *MIS quarterly*, pages 1165–1188, 2012.

[16] Ahmed Bader, Hakim Ghazzai, Abdullah Kadri, and Mohamed-Slim Alouini. Front-end intelligence for large-scale application-oriented internet-of-things. *IEEE Access*, 4:3257–3272, 2016.

[17] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9(2):48–57, 2014.

[18] Gema Bello-Orgaz, Jason J Jung, and David Camacho. Social big data: Recent achievements and new challenges. *Information Fusion*, 28:45–59, 2016.

[19] Stephen Kaisler, Frank Armour, J Alberto Espinosa, and William Money. Big data: Issues and challenges moving forward. In *System sciences (HICSS), 2013 46th Hawaii international conference on*, pages 995–1004. IEEE, 2013.

[20] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.

[21] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[22] David Lowe. Adaptive radial basis function nonlinearities, and the problem of generalisation. In *Artificial Neural Networks, 1989., First IEE International Conference on (Conf. Publ. No. 313)*, pages 171–175. IET, 1989.

[23] Yoh-Han Pao, Gwang-Hoon Park, and Dejan J Sobajic. Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing*, 6 (2):163–180, 1994.

[24] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: a new learning scheme of feedforward neural networks. In *Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on*, volume 2, pages 985–990. IEEE, 2004.

[25] Paolo Gastaldo, Rodolfo Zunino, Erik Cambria, and Sergio Decherchi. Combining elm with random projections. *IEEE intelligent systems*, 28(6):46–48, 2013.

[26] Ali Rahimi and Benjamin Recht. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pages 1313–1320, 2009.

[27] Maria-Florina Balcan, Avrim Blum, and Nathan Srebro. A theory of learning with similarity functions. *Machine Learning*, 72(1-2):89–112, 2008.

[28] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[30] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[31] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[32] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, 17(4):879–892, 2006.

[33] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.

[34] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[35] Luca Oneto, Federica Bisio, Erik Cambria, and Davide Anguita. Statistical learning theory and elm for big social data analysis. *ieee CompUTATionAl inTelliGenCe mAGAzine*, 11(3):45–55, 2016.

[36] John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[37] Paolo Gastaldo, Federica Bisio, Sergio Decherchi, and Rodolfo Zunino. Sim-elm: Connecting the elm model with similarity-function learning. *Neural Networks*, 74: 22–34, 2016.

[38] Xia Liu, Shaobo Lin, Jian Fang, and Zongben Xu. Is extreme learning machine feasible? a theoretical assessment (part i). *IEEE Transactions on Neural Networks and Learning Systems*, 26(1):7–20, 2015.

[39] Shaobo Lin, Xia Liu, Jian Fang, and Zongben Xu. Is extreme learning machine feasible? a theoretical assessment (part ii). *IEEE Transactions on Neural Networks and Learning Systems*, 26(1):21–34, 2015.

[40] Yoan Miche, Antti Sorjamaa, Patrick Bas, Olli Simula, Christian Jutten, and Amaury Lendasse. Op-elm: optimally pruned extreme learning machine. *IEEE transactions on neural networks*, 21(1):158–162, 2010.

[41] Sergio Decherchi, Paolo Gastaldo, Alessio Leoncini, and Rodolfo Zunino. Efficient digital implementation of extreme learning machines for classification. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(8):496–500, 2012.

[42] Jiuwen Cao, Zhiping Lin, and Guang-Bin Huang. Self-adaptive evolutionary extreme learning machine. *Neural processing letters*, 36(3):285–305, 2012.

[43] You Xu and Yang Shu. Evolutionary extreme learning machine–based on particle swarm optimization. In *International Symposium on Neural Networks*, pages 644–652. Springer, 2006.

[44] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[45] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*, 18:187–1, 2017.

[46] Guang-Bin Huang, Qin-Yu Zhu, KZ Mao, Chee-Kheong Siew, Paramasivan Saratchandran, and Narasimhan Sundararajan. Can threshold networks be trained directly? *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(3): 187–191, 2006.

[47] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[48] Uci datasets [online] available. URL `https://archive.ics.uci.edu/ml/datasets.html`.

[49] Davide Anguita, Alessandro Ghio, Luca Oneto, and Sandro Ridella. In-sample model selection for trimmed hinge loss support vector machine. *Neural processing letters*, 36(3):275–283, 2012.

[50] Davide Anguita, Alessandro Ghio, Luca Oneto, and Sandro Ridella. In-sample and out-of-sample model selection and error estimation for support vector machines. *IEEE Transactions on Neural Networks and Learning Systems*, 23(9):1390–1406, 2012.

[51] Leo Breiman. Some properties of splitting criteria. *Machine Learning*, 24(1):41–47, 1996.

[52] Ivor W Tsang, James T Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(Apr):363–392, 2005.

[53] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[54] Forest covertype, [online] available. URL `http://kdd.ics.uci.edu/databases/covertype/covertype.htm`.

[55] Susy dataset [online] available. URL `https://archive.ics.uci.edu/ml/datasets/SUSY`.

[56] Higgs dataset [online] available. URL `https://archive.ics.uci.edu/ml/datasets/HIGGS`.

[57] Iman Sharafaldin, A Habibi Lashkari, and Ali A Ghorbani. Toward generating a new intrusion detection dataset and intrusion traffic characterization. In *Proceedings of fourth international conference on information systems security and privacy, ICISSP*, 2018.

[58] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[59] Tianqi Wu, Min Yao, and Jianhua Yang. Dolphin swarm extreme learning machine. *Cognitive Computation*, 9(2):275–284, 2017.

[60] Xiaowei Xue, Min Yao, Zhaohui Wu, and Jianhua Yang. Genetic ensemble of extreme learning machine. *Neurocomputing*, 129:175–184, 2014.

[61] Hui-yuan Tian, Shi-jian Li, Tian-qi Wu, and Min Yao. An extreme learning machine based on artificial immune system. In *The 8th International Conference on Extreme Learning Machines (ELM2017), Yantai, China*, 2017.

[62] Aakash Patil, Shanlan Shen, Enyi Yao, and Arindam Basu. Hardware architecture for large parallel array of random feature extractors applied to image recognition. *Neurocomputing*, 261:193–203, 2017.

[63] Yi Chen, Enyi Yao, and Arindam Basu. A 128 channel 290 gmacs/w machine learning based co-processor for intention decoding in brain machine interfaces. In *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on*, pages 3004–3007. IEEE, 2015.

[64] Spyridon Plakias and Efstathios Stamatatos. Tensor space models for authorship identification. In *Hellenic Conference on Artificial Intelligence*, pages 239–249. Springer, 2008.

[65] Paolo Gastaldo, Luigi Pinna, Lucia Seminara, Maurizio Valle, and Rodolfo Zunino. A tensor-based pattern-recognition framework for the interpretation of touch modality in artificial skin systems. *IEEE Sensors Journal*, 14(7):2216–2225, 2014.

[66] Xiao-Wei Wang, Dan Nie, and Bao-Liang Lu. Emotional state classification from eeg data using machine learning approach. *Neurocomputing*, 129:94–106, 2014.

[67] Qibin Zhao, Guoxu Zhou, Tulay Adali, Liqing Zhang, and Andrzej Cichocki. Kernelization of tensor-based models for multiway data analysis: Processing of multidimensional structured data. *IEEE Signal Processing Magazine*, 30(4):137–148, 2013.

[68] Qibin Zhao, Liqing Zhang, and Andrzej Cichocki. Multilinear and nonlinear generalizations of partial least squares: an overview of recent advances. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(2):104–115, 2014.

[69] Yang Liu, Yan Liu, and Keith CC Chan. Tensor distance based multilinear locality-preserved maximum information embedding. *IEEE Transactions on Neural Networks*, 21(11):1848–1854, 2010.

[70] Tohru Iwasaki and Tetsuo Furukawa. Tensor som and tensor gtm: Nonlinear tensor analysis by topographic mappings. *Neural Networks*, 77:107–125, 2016.

[71] Zhihui Lai, Wai Keung Wong, Yong Xu, Cairong Zhao, and Mingming Sun. Sparse alignment for robust tensor learning. *IEEE transactions on neural networks and learning systems*, 25(10):1779–1792, 2014.

[72] Haiping Lu, Konstantinos N Plataniotis, and Anastasios N Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, 44 (7):1540–1551, 2011.

[73] Xi'ai Chen, Zhi Han, Yao Wang, Qian Zhao, Deyu Meng, Lin Lin, and Yandong Tang. A generalized model for robust tensor factorization with noise modeling by mixture of gaussians. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[74] Parinya Sanguansat. Higher-order random projection for tensor object recognition. In *Communications and Information Technologies (ISCIT), 2010 International Symposium on*, pages 615–619. IEEE, 2010.

[75] Marco Signoretto, Quoc Tran Dinh, Lieven De Lathauwer, and Johan AK Suykens. Learning with tensors: a framework based on convex optimization and spectral regularization. *Machine Learning*, 94(3):303–351, 2014.

[76] Dacheng Tao, Xuelong Li, Weiming Hu, Stephen Maybank, and Xindong Wu. Supervised tensor learning. In *Data Mining, Fifth IEEE International Conference on*, pages 8–pp. IEEE, 2005.

[77] Xian Guo, Xin Huang, Lefei Zhang, Liangpei Zhang, Antonio Plaza, and Jón Atli Benediktsson. Support tensor machines for classification of hyperspectral remote sensing imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 54(6): 3248–3264, 2016.

[78] Kishan Wimalawarne, Ryota Tomioka, and Masashi Sugiyama. Theoretical and experimental analyses of tensor-based regression and classification. *Neural computation*, 28(4):686–715, 2016.

[79] Zhifeng Hao, Lifang He, Bingqian Chen, and Xiaowei Yang. A linear support higher-order tensor machine for classification. *IEEE Transactions on Image Processing*, 22(7):2911–2920, 2013.

[80] Yi Xiang, Qian Jiang, Jing He, Xin Jin, LiWen Wu, and Shaowen Yao. The advance of support tensor machine. In *2018 IEEE 16th International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 121–128. IEEE, 2018.

[81] Marco Signoretto, Lieven De Lathauwer, and Johan AK Suykens. A kernel-based framework to tensorial data analysis. *Neural networks*, 24(8):861–874, 2011.

[82] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4): 1253–1278, 2000.

[83] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, 2015.

[84] THJM Peeters, PR Rodrigues, A Vilanova, and BM ter Haar Romeny. Analysis of distance/similarity measures for diffusion tensor imaging. In *Visualization and Processing of Tensor Fields*, pages 113–136. Springer, 2009.

[85] Thomas K Landauer. *Latent semantic analysis*. Wiley Online Library, 2006.

[86] Richard A Harshman. Foundations of the parafac procedure: Models and conditions for an" explanatory" multimodal factor analysis. *Working Papers in Phonetics*, 1970.

[87] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[88] Bastian Leibe and Bernt Schiele. Analyzing appearance and contour based methods for object categorization. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–409. IEEE, 2003.

[89] Peter N. Belhumeur, João P Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):711–720, 1997.

[90] M-E Nilsback and Andrew Zisserman. A visual vocabulary for flower classification. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1447–1454. IEEE, 2006.

[91] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004.

[92] Tae-Kyun Kim and Roberto Cipolla. Canonical correlation analysis of video volume tensors for action categorization and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(8):1415–1428, 2009.

[93] Irene Rodriguez-Lujan, Jordi Fonollosa, Alexander Vergara, Margie Homer, and Ramon Huerta. On the calibration of sensor arrays for pattern recognition using the minimal number of experiments. *Chemometrics and Intelligent Laboratory Systems*, 130:123–134, 2014.

[94] Deng Cai, Xiaofei He, Yuxiao Hu, Jiawei Han, and Thomas Huang. Learning a spatially smooth subspace for face recognition. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–7. IEEE, 2007.

[95] Rahat Iqbal, Faiyaz Doctor, Brian More, Shahid Mahmud, and Usman Yousuf. Big data analytics and computational intelligence for cyber–physical systems: Recent trends and state of the art applications. *Future Generation Computer Systems*, 2017.

[96] Chin-Teng Lin, Yu-Ting Liu, Shang-Lin Wu, Zehong Cao, Yu-Kai Wang, Chih-Sheng Huang, Jung-Tai King, Shi-An Chen, Shao-Wei Lu, and Chun-Hsiang Chuang. Eeg-based brain-computer interfaces: A novel neurotechnology and computational intelligence method. *IEEE Systems, Man, and Cybernetics Magazine*, 3(4):16–26, 2017.

[97] Paolo Gastaldo, Luigi Pinna, Lucia Seminara, Maurizio Valle, and Rodolfo Zunino. A tensor-based approach to touch modality classification by using machine learning. *Robotics and Autonomous Systems*, 63:268–278, 2015.

[98] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L Reyes-Ortiz. Human activity recognition on smartphones using a multiclass hardware-friendly support vector machine. In *International workshop on ambient assisted living*, pages 216–223. Springer, 2012.

[99] Arindam Basu, Sun Shuo, Hongming Zhou, Meng Hiot Lim, and Guang-Bin Huang. Silicon spiking neurons for hardware implementation of extreme learning machines. *Neurocomputing*, 102:125–134, 2013.

[100] Jose V Frances-Villora, A Rosado-Muñoz, José M Martínez-Villena, Manuel Bataller-Mompean, Juan Fco Guerrero, and Marek Wegrzyn. Hardware implementation of real-time extreme learning machine in fpga: analysis of precision, resource occupation and performance. *Computers & Electrical Engineering*, 51: 139–156, 2016.

[101] Alfredo Canziani, Eugenio Culurciello, and Adam Paszke. Evaluation of neural network architectures for embedded systems. In *Circuits and Systems (ISCAS), 2017 IEEE International Symposium on*, pages 1–4. IEEE, 2017.

[102] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[103] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[104] Enyi Yao and Arindam Basu. Vlsi extreme learning machine: A design space exploration. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(1):60–74, 2017.

[105] Chang-Hung Tsai, Yu-Ting Chih, Wing Hung Wong, and Chen-Yi Lee. A hardware-efficient sigmoid function with adjustable precision for a neural network system. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(11): 1073–1077, 2015.

[106] Makoto Murase. Linear feedback shift register, February 18 1992. US Patent 5,090,035.

[107] Linear feedback shift register maximal length table. URL `https://www.xilinx.com/support/documentation/application_notes/xapp052.pdf`.

[108] Erik Cambria, Soujanya Poria, Alexander Gelbukh, and Mike Thelwall. Sentiment analysis is a big suitcase. *IEEE Intelligent Systems*, 32(6):74–80, 2017.

[109] Soujanya Poria, Erik Cambria, and Alexander Gelbukh. Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based Systems*, 108:42–49, 2016.

[110] Yukun Ma, Erik Cambria, and Sa Gao. Label embedding for zero-shot fine-grained named entity typing. In *COLING*, pages 171–180, Osaka, 2016.

[111] Dheeraj Rajagopal, Erik Cambria, Daniel Olsher, and Kenneth Kwok. A graph-based approach to commonsense concept extraction and semantic similarity detection. In *WWW*, pages 565–570, Rio De Janeiro, 2013.

[112] Soujanya Poria, Erik Cambria, D Hazarika, and Prateek Vij. A deeper look into sarcastic tweets using deep convolutional neural networks. In *COLING*, pages 1601–1612, 2016.

[113] Soujanya Poria, Iti Chaturvedi, Erik Cambria, and Amir Hussain. Convolutional MKL based multimodal emotion recognition and sentiment analysis. In *ICDM*, pages 439–448, Barcelona, 2016.

[114] Claudia Meda, Edoardo Ragusa, Christian Gianoglio, Rodolfo Zunino, Augusto Ottaviano, Eugenio Scillia, and Roberto Surlinelli. Spam detection of twitter traffic: A framework based on random forests and non-uniform feature sampling. In *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*, pages 811–817. IEEE, 2016.

[115] Marjan Van de Kauter, Diane Breesch, and Véronique Hoste. Fine-grained analysis of explicit and implicit sentiment in financial news articles. *Expert Systems with Applications*, 42(11):4999 – 5010, 2015.

[116] Sven Rill, Dirk Reinel, Jörg Scheidt, and Roberto V. Zicari. Politwi: Early detection of emerging political topics on twitter and the impact on concept-level sentiment analysis. *Knowledge-Based Systems*, 69(0):24 – 33, 2014.

[117] Prakhar Biyani, Sumit Bhatia, Cornelia Caragea, and Prasenjit Mitra. Using non-lexical features for identifying factual and opinionative threads in online forums. *Knowledge-Based Systems*, 69(0):170 – 178, 2014.

[118] Iti Chaturvedi, Erik Cambria, and David Vilares. Lyapunov filtering of objectivity for Spanish sentiment model. In *IJCNN*, pages 4474–4481, Vancouver, 2016.

[119] Theresa Wilson. *Fine-grained Subjectivity and Sentiment Analysis: Recognizing the Intensity, Polarity, and Attitudes of private states.* PhD thesis, Intelligent Systems Program, University of Pittsburgh, 2007.

[120] Srikumar Krishnamoorthy. Linguistic features for review helpfulness prediction. *Expert Systems with Applications*, 42(7):3751 – 3759, 2015.

[121] Iti Chaturvedi, Erik Cambria, Feida Zhu, Lin Qiua, and Wee k Ng. Multilingual subjectivity detection using deep multiple kernel learning. *Proceedings of KDD, Sydney*, 2015.

[122] Sadanori Konishi, Tomohiro Ando, and Seiya Imoto. Bayesian information criteria and smoothing parameter selection in radial basis function networks. *Biometrika*, 91(1):27–43, 2004. doi: 10.1093/biomet/91.1.27.

[123] Rada Mihalcea, Carmen Banea, and Janyce Wiebe. Learning multilingual subjective language via cross-lingual projections. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 976–983. Association for Computational Linguistics, 2007.

[124] Janyce Wiebe and Ellen Riloff. Creating subjective and objective sentence classifiers from unannotated texts. In *Proceedings of the 6th International Conference on Computational Linguistics and Intelligent Text Processing*, pages 486–497, 2005.

[125] Louis-Philippe Morency, Rada Mihalcea, and Payal Doshi. Towards multimodal sentiment analysis: Harvesting opinions from the web. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 169–176. ACM, 2011.

[126] P. Baranyi, Kin-Fong Lei, and Yeung Yam. Complexity reduction of singleton based neuro-fuzzy algorithm. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 4, pages 2503–2508 vol.4, 2000.

[127] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70(1):489–501, 2006.

[128] Jiahua Luo, Chi-Man Vong, and Pak-Kin Wong. Sparse bayesian extreme learning machine for multi-classification. *IEEE Transactions on Neural Networks and Learning Systems*, 25(4):836–843, 2014.

[129] Verónica Pérez-Rosas, Rada Mihalcea, and Louis-Philippe Morency. Utterance-level multimodal sentiment analysis. In *ACL (1)*, pages 973–982, 2013.

[130] Ellen Riloff and Janyce Wiebe. Learning extraction patterns for subjective expressions. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 105–112. Association for Computational Linguistics, 2003.

[131] Rada Mihalcea, Carmen Banea, and Janyce Wiebe. Learning multilingual subjective language via cross-lingual projections. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 976–983. Association for Computational Linguistics, June 2007.

[132] Julio Villena-Román, Janine García-Morera, Miguel Ángel García Cumbreras, Eugenio Martínez-Cámara, Maria Teresa Martín-Valdivia, and Luis Alfonso Ureña López. Overview of tass 2015. In *Proceedings of TASS 2015: Workshop on Sentiment Analysis at SEPLN*, 2015.

[133] David Vilares, Yerai Doval, Miguel A Alonso, and Carlos Gómez-Rodríguez. Lys at tass 2014: A prototype for extracting and analysing aspects from spanish tweets. *Proceedings of the TASS workshop at SEPLN*, 2014.

[134] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 655–665. Association for Computational Linguistics, 2014.

[135] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[136] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[137] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[138] Andriy Mnih and Geoffrey Hinton. Three new graphical models for statistical language modelling. In *Proceedings of the 24th international conference on Machine learning*, pages 641–648. ACM, 2007.

[139] Jian Tang, Meng Qu, and Qiaozhu Mei. Pte: Predictive text embedding through large-scale heterogeneous text networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1165–1174. ACM, 2015.

[140] Suhang Wang, Jiliang Tang, Charu Aggarwal, and Huan Liu. Linked document embedding for classification. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 115–124. ACM, 2016.

[141] Yukun Ma, Haiyun Peng, and Erik Cambria. Targeted aspect-based sentiment analysis via embedding commonsense knowledge into an attentive LSTM. In *AAAI*, pages 5876–5883, 2018.

[142] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL `http://www.aclweb.org/anthology/D14-1162`.

[143] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

[144] Saif M Mohammad and Peter D Turney. Crowdsourcing a word–emotion association lexicon. *Computational Intelligence*, 29(3):436–465, 2013.

[145] Erik Cambria, Soujanya Poria, Devamanyu Hazarika, and Kenneth Kwok. SenticNet 5: Discovering conceptual primitives for sentiment analysis by means of context embeddings. In *AAAI*, pages 1795–1802, 2018.

[146] Yang Li, Quan Pan, Tao Yang, Suhang Wang, Jiliang Tang, and Erik Cambria. Learning word representations for sentiment analysis. *Cognitive Computation*, 9 (6):843–851, 2017.

[147] Xiaodong Li, Haoran Xie, Li Chen, Jianping Wang, and Xiaotie Deng. News impact on stock price return via sentiment analysis. *Knowledge-Based Systems*, 69:14–23, 2014.

[148] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[149] Shusen Liu, Dan Maljovec, Bei Wang, Peer-Timo Bremer, and Valerio Pascucci. Visualizing high-dimensional data: Advances in the past decade. *IEEE transactions on visualization and computer graphics*, 23(3):1249–1268, 2017.

[150] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[151] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, pages 583–588. Springer, 1997.

[152] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[153] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[154] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[155] Marco. J. Ferrarotti, Walter Rocchia, and Sergio Decherchi. Finding principal paths in data space. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–14, 2018. ISSN 2162-237X. doi: 10.1109/TNNLS.2018.2884792.

[156] Robert Plutchik. The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. *American scientist*, 89(4):344–350, 2001.

[157] Erik Cambria, Andrew Livingstone, and Amir Hussain. The hourglass of emotions. In *Cognitive behavioural systems*, pages 144–157. Springer, 2012.

[158] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *Advances in neural information processing systems*, pages 585–592, 1995.

[159] Hugo Liu and Push Singh. Conceptnet—a practical commonsense reasoning toolkit. *BT technology journal*, 22(4):211–226, 2004.

[160] Carlo Strapparava, Alessandro Valitutti, et al. Wordnet affect: an affective extension of wordnet. In *Lrec*, volume 4, pages 1083–1086. Citeseer, 2004.

[161] Erik Cambria and Amir Hussain. *Sentic Computing: A Common-Sense-Based Framework for Concept-Level Sentiment Analysis*. Springer, Cham, Switzerland, 2015.

[162] Mohammad Soleymani, David Garcia, Brendan Jou, Björn Schuller, Shih-Fu Chang, and Maja Pantic. A survey of multimodal sentiment analysis. *Image and Vision Computing*, 65:3–14, 2017.

[163] Soujanya Poria, Erik Cambria, Rajiv Bajpai, and Amir Hussain. A review of affective computing: From unimodal analysis to multimodal fusion. *Information Fusion*, 37:98–125, 2017.

[164] Devamanyu Hazarika, Soujanya Poria, Amir Zadeh, Erik Cambria, Louis-Philippe Morency, and Roger Zimmermann. Conversational memory network for emotion recognition in dyadic dialogue videos. In *NAACL*, pages 2122–2132, 2018.

[165] Erik Cambria, Amir Hussain, Catherine Havasi, and Chris Eckl. SenticSpace: Visualizing opinions and sentiments in a multi-dimensional vector space. In Rossitza Setchi, Ivan Jordanov, Robert Howlett, and Lakhmi Jain, editors, *Knowledge-Based and Intelligent Information and Engineering Systems*, volume 6279 of *Lecture Notes in Artificial Intelligence*, pages 385–393. Springer, Berlin, 2010.

[166] Erik Cambria, Daniel Olsher, and Kenneth Kwok. Sentic activation: A two-level affective common sense reasoning framework. In *AAAI*, pages 186–192, Toronto, 2012.

[167] Erik Cambria, Amir Hussain, Tariq Durrani, Catherine Havasi, Chris Eckl, and James Munro. Sentic computing for patient centered application. In *IEEE ICSP*, pages 1279–1282, 2010.

[168] Erik Cambria. An introduction to concept-level sentiment analysis. In Félix Castro, Alexander Gelbukh, and Miguel González, editors, *Advances in Soft Computing and Its Applications*, volume 8266 of *Lecture Notes in Computer Science*, pages 478–483, Berlin, 2013. Springer-Verlag.

[169] Can Xu, Suleyman Cetintas, Kuang-Chih Lee, and Li-Jia Li. Visual sentiment prediction with deep convolutional neural networks. *arXiv preprint arXiv:1411.5731*, 2014.

[170] Quanzeng You, Jiebo Luo, Hailin Jin, and Jianchao Yang. Robust image sentiment analysis using progressively trained and domain transferred deep networks. In *AAAI*, pages 381–388, 2015.

[171] Jyoti Islam and Yanqing Zhang. Visual sentiment analysis for social images using transfer learning approach. In *Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloud-SocialCom-SustainCom), 2016 IEEE International Conferences on*, pages 124–130. IEEE, 2016.

[172] Victor Campos, Brendan Jou, and Xavier Giro-i Nieto. From pixels to sentiment: Fine-tuning cnns for visual sentiment prediction. *Image and Vision Computing*, 65:15–22, 2017.

[173] Jiebo Luo, Damian Borth, and Quanzeng You. Social multimedia sentiment analysis. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 1953–1954. ACM, 2017.

[174] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[175] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[176] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[177] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[178] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[179] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[180] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[181] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al.

Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[182] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[183] Wang Wei-ning, Yu Ying-lin, and Jiang Sheng-ming. Image retrieval by emotional semantics: A study of emotional space and feature extraction. In *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, volume 4, pages 3534–3539. IEEE, 2006.

[184] Stefan Siersdorfer, Enrico Minack, Fan Deng, and Jonathon Hare. Analyzing and predicting sentiment of images on the social web. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 715–718. ACM, 2010.

[185] Jana Machajdik and Allan Hanbury. Affective image classification using features inspired by psychology and art theory. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 83–92. ACM, 2010.

[186] Victoria Yanulevskaya, Jan C van Gemert, Katharina Roth, Ann-Katrin Herbold, Nicu Sebe, and Jan-Mark Geusebroek. Emotional valence categorization using holistic image features. In *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pages 101–104. IEEE, 2008.

[187] Kuan-Chuan Peng, Tsuhan Chen, Amir Sadovnik, and Andrew C Gallagher. A mixed bag of emotions: Model, predict, and transfer emotion distributions. In *CVPR*, pages 860–868, 2015.

[188] Jianbo Yuan, Sean Mcdonough, Quanzeng You, and Jiebo Luo. Sentribute: image sentiment analysis from a mid-level perspective. In *Proceedings of the Second International Workshop on Issues of Sentiment Discovery and Opinion Mining*, page 10. ACM, 2013.

[189] Damian Borth, Rongrong Ji, Tao Chen, Thomas Breuel, and Shih-Fu Chang. Large-scale visual sentiment ontology and detectors using adjective noun pairs. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 223–232. ACM, 2013.

[190] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[191] Tao Chen, Felix X Yu, Jiawei Chen, Yin Cui, Yan-Ying Chen, and Shih-Fu Chang. Object-based visual sentiment concept analysis and application. In *Proceedings*

of the 22nd ACM international conference on Multimedia, pages 367–376. ACM, 2014.

[192] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.

[193] Victor Campos, Amaia Salvador, Xavier Giro-i Nieto, and Brendan Jou. Diving deep into sentiment: Understanding fine-tuned cnns for visual sentiment prediction. In *Proceedings of the 1st International Workshop on Affect & Sentiment in Multimedia*, pages 57–62. ACM, 2015.

[194] Tao Chen, Damian Borth, Trevor Darrell, and Shih-Fu Chang. Deepsentibank: Visual sentiment concept classification with deep convolutional neural networks. *arXiv preprint arXiv:1410.8586*, 2014.

[195] Brendan Jou, Tao Chen, Nikolaos Pappas, Miriam Redi, Mercan Topkara, and Shih-Fu Chang. Visual affect around the world: A large-scale multilingual visual sentiment ontology. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 159–168. ACM, 2015.

[196] Shaojing Fan, Ming Jiang, Zhiqi Shen, Bryan L Koenig, Mohan S Kankanhalli, and Qi Zhao. The role of visual attention in sentiment prediction. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 217–225. ACM, 2017.

[197] Delia Fernandez, Alejandro Woodward, Víctor Campos, Xavier Giró-i Nieto, Brendan Jou, and Shih-Fu Chang. More cat than cute?: Interpretable prediction of adjective-noun pairs. In *Proceedings of the Workshop on Multimodal Understanding of Social, Affective and Subjective Attributes*, pages 61–69. ACM, 2017.

[198] Takuya Narihira, Damian Borth, Stella X Yu, Karl Ni, and Trevor Darrell. Mapping images to sentiment adjective noun pairs with factorized neural nets. *arXiv preprint arXiv:1511.06838*, 2015.

[199] Jingwen Wang, Jianlong Fu, Yong Xu, and Tao Mei. Beyond object recognition: Visual sentiment analysis with deep coupled adjective and noun neural networks. In *IJCAI*, pages 3484–3490, 2016.

[200] Brendan Jou and Shih-Fu Chang. Deep cross residual learning for multitask visual recognition. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 998–1007. ACM, 2016.

[201] Honglin Zheng, Tianlang Chen, Quanzeng You, and Jiebo Luo. When saliency meets sentiment: Understanding how image content invokes emotion and sentiment. In *Image Processing (ICIP), 2017 IEEE International Conference on*, pages 630–634. IEEE, 2017.

[202] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.

[203] Ming Sun, Jufeng Yang, Kai Wang, and Hui Shen. Discovering affective regions in deep convolutional neural networks for visual sentiment prediction. In *Multimedia and Expo (ICME), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[204] Quanzeng You, Hailin Jin, and Jiebo Luo. Visual sentiment analysis by attending on local image regions. In *AAAI*, pages 231–237, 2017.

[205] Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[206] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[207] Alexander Patrick Mathews, Lexing Xie, and Xuming He. Senticap: Generating image descriptions with sentiments. In *AAAI*, pages 3574–3580, 2016.

[208] Andrew Shin, Yoshitaka Ushiku, and Tatsuya Harada. Image captioning with sentiment terms via weakly-supervised sentiment dataset. In *BMVC*, 2016.

[209] Quanzeng You, Hailin Jin, and Jiebo Luo. Image captioning at will: A versatile scheme for effectively injecting sentiments into image descriptions. *arXiv preprint arXiv:1801.10121*, 2018.

[210] Tushar Karayil, Philipp Blandfort, Damian Borth, and Andreas Dengel. Generating affective captions using concept and syntax transition networks. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1111–1115. ACM, 2016.

[211] Yan Sun and Bo Ren. Automatic image description generation with emotional classifiers. In *CCF Chinese Conference on Computer Vision*, pages 748–763. Springer, 2017.

[212] Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. Multimodal semi-supervised learning for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 902–909. IEEE, 2010.

[213] Marie Katsurai and Shin'ichi Satoh. Image sentiment analysis using latent correlations among visual, textual, and sentiment views. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2837–2841. IEEE, 2016.

[214] Yunchao Gong, Qifa Ke, Michael Isard, and Svetlana Lazebnik. A multi-view embedding space for modeling internet images, tags, and their semantics. *International journal of computer vision*, 106(2):210–233, 2014.

[215] Guoyong Cai and Binbin Xia. Convolutional neural networks for multimedia sentiment analysis. In *Natural Language Processing and Chinese Computing*, pages 159–167. Springer, 2015.

[216] Quanzeng You. Sentiment and emotion analysis for social multimedia: methodologies and applications. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1445–1449. ACM, 2016.

[217] Quanzeng You, Jiebo Luo, Hailin Jin, and Jianchao Yang. Joint visual-textual sentiment analysis with deep neural networks. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 1071–1074. ACM, 2015.

[218] Xingyue Chen, Yunhong Wang, and Qingjie Liu. Visual and textual sentiment analysis using deep fusion convolutional neural networks. *arXiv preprint arXiv:1711.07798*, 2017.

[219] Sicheng Zhao, Guiguang Ding, Yue Gao, and Jungong Han. Learning visual emotion distributions via multi-modal features fusion. In *Proceedings of the 2017 ACM on Multimedia Conference*, pages 369–377. ACM, 2017.

[220] Jufeng Yang, Dongyu She, and Ming Sun. Joint image emotion classification and distribution learning via deep convolutional neural network. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 3266–3272. AAAI Press, 2017.

[221] Evangelos Sariyanidi, Hatice Gunes, and Andrea Cavallaro. Automatic analysis of facial affect: A survey of registration, representation, and recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(6):1113–1133, 2015.

[222] Sebastian Kaltwang, Sinisa Todorovic, and Maja Pantic. Doubly sparse relevance vector machine for continuous facial behavior estimation. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1748–1761, 2016.

[223] Robert Walecki, Ognjen Rudovic, Vladimir Pavlovic, and Maja Pantic. Copula ordinal regression for joint estimation of facial action unit intensity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4902–4910, 2016.

[224] Ognjen Rudovic, Vladimir Pavlovic, and Maja Pantic. Context-sensitive dynamic ordinal regression for intensity estimation of facial action units. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):944–958, 2015.

[225] Yuchen Wu, Jianbo Yuan, Quanzeng You, and Jiebo Luo. The effect of pets on happiness: a data-driven approach via large-scale social media. In *Big Data (Big Data), 2016 IEEE International Conference on*, pages 1889–1894. IEEE, 2016.

[226] Vasileios Belagiannis and Andrew Zisserman. Recurrent human pose estimation. In *Automatic Face & Gesture Recognition (FG 2017), 2017 12th IEEE International Conference on*, pages 468–475. IEEE, 2017.

[227] Abhinav Dhall, Jyoti Joshi, Karan Sikka, Roland Goecke, and Nicu Sebe. The more the merrier: Analysing the affect of a group of people in images. In *Automatic Face and Gesture Recognition (FG), 2015 11th IEEE International Conference and Workshops on*, volume 1, pages 1–8. IEEE, 2015.

[228] Peter J Lang, Margaret M Bradley, and Bruce N Cuthbert. International affective picture system (iaps): Technical manual and affective ratings. *NIMH Center for the Study of Emotion and Attention*, pages 39–58, 1997.

[229] Quanzeng You, Jiebo Luo, Hailin Jin, and Jianchao Yang. Building a large scale dataset for image emotion recognition: The fine print and the benchmark. In *AAAI*, pages 308–314, 2016.

[230] Jufeng Yang, Ming Sun, and Xiaoxiao Sun. Learning visual sentiment distributions via augmented conditional probability neural network. In *AAAI*, pages 224–230, 2017.

[231] D Ramyachitra and P Manikandan. Imbalanced dataset classification and solutions: a review. *International Journal of Computing and Business Research (IJCBR)*, 5(4), 2014.