



UNICAMP

UNIVERSIDADE ESTADUAL DE
CAMPINAS

Instituto de Matemática, Estatística e
Computação Científica

DAVID ERNESTO CARO CONTRERAS

**Toward Deep Neural Networks with Generalized
Morphological Components**

**Rumo a Redes Neurais Profundas com
Componentes Morfológicos Generalizados**

Campinas

2020

David Ernesto Caro Contreras

**Toward Deep Neural Networks with Generalized
Morphological Components**

**Rumo a Redes Neurais Profundas com Componentes
Morfológicos Generalizados**

Tese apresentada ao Instituto de Matemática, Estatística e Computação Científica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Doutor em Matemática Aplicada.

Thesis presented to the Institute of Mathematics, Statistics and Scientific Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Doctor in Applied Mathematics.

Supervisor: Peter Sussner

Este exemplar corresponde à versão final da Tese defendida pelo aluno David Ernesto Caro Contreras e orientada pelo Prof. Dr. Peter Sussner.

Campinas

2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

C22t Caro Contreras, David Ernesto, 1983-
Toward deep neural networks with generalized morphological components /
David Ernesto Caro Contreras. – Campinas, SP : [s.n.], 2020.

Orientador: Peter Sussner.

Tese (doutorado) – Universidade Estadual de Campinas, Instituto de
Matemática, Estatística e Computação Científica.

1. Aprendizado profundo. 2. Redes neurais convolucionais. 3. Morfologia
matemática. 4. Aprendizagem supervisionada (Aprendizado do computador). I.
Sussner, Peter, 1961-. II. Universidade Estadual de Campinas. Instituto de
Matemática, Estatística e Computação Científica. III. Título.

Informações para Biblioteca Digital

Título em outro idioma: Rumo a redes neurais profundas com componentes morfológicos generalizados

Palavras-chave em inglês:

Deep learning

Convolutional neural networks

Mathematical morphology

Supervised learning (Machine learning)

Área de concentração: Matemática Aplicada

Titulação: Doutor em Matemática Aplicada

Banca examinadora:

Peter Sussner [Orientador]

João Batista Florindo

Fernando José Von Zuben

Juan Humberto Sossa Azuela

Gerhard X. Ritter

Data de defesa: 29-05-2020

Programa de Pós-Graduação: Matemática Aplicada

Identificação e informações acadêmicas do(a) aluno(a)

- ORCID do autor: <https://orcid.org/0000-0003-1969-4092>

- Currículo Lattes do autor: <http://lattes.cnpq.br/3624003341116251>

**Tese de Doutorado defendida em 29 de maio de 2020 e aprovada
pela banca examinadora composta pelos Profs. Drs.**

Prof(a). Dr(a). PETER SUSSNER

Prof(a). Dr(a). JOÃO BATISTA FLORINDO

Prof(a). Dr(a). FERNANDO JOSÉ VON ZUBEN

Prof(a). Dr(a). JUAN HUMBERTO SOSSAAZUELA

Prof(a). Dr(a). GERHARD X. RITTER

A Ata da Defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria de Pós-Graduação do Instituto de Matemática, Estatística e Computação Científica.

Acknowledgements

Agradezco infinitamente a mi familia, especialmente a mis padres, a mi Mama I y a mis hermanos por su constante apoyo y cariño.

A mi amor Noemi que a mi lado recorre el camino, empero no existan palabras en el lenguaje popular, es mi Crux cuando estoy perdido, el mejor de los sueños cuando no estoy dormido y mi alegría al despertar.

A Eli, a Beatriz y a la profesora Maria Luisa que se volvieron mi familia brasileña y mi soporte en los momentos más difíciles.

A mis compañeros y amigos de la UNICAMP: Priscila, Fidelis, Fernanda Rocha, Israel, Alejandro, Fabio, Brunna, Aline, Dani, Fito, Nil, Rafael, Fernanda Bia, Silvia, Lis, Geizane, Vanterler, Francis, Jenny, Luana, Marcia, Cristina, Vinicius, Ricardo, Aline, Majid, Fernanda Teixeira, Ciro, João, Renata, Elaine, Rebeca, Anderson, Melissa, Pam, Kiwi, Thiago, Pau, Silvio, Chrislen, Isabelly, Leticia y a todas las personas con los que compartí café, largas noches de estudios y fiestas. A los grupos de investigación de MiLab y Bio-matemática con los que convivía y reía todos los días.

A mis amigos en Guadalajara, que continúan conmigo aún en la distancia.

A los profesores del IMECC que con paciencia y sapiencia comparten su conocimiento y amor por la ciencia. Especialmente a los Profesores Wilson Castro Ferreira Jr, Laécio, Marcos Valle por su dedicación como maestros. A los Profesores Fernando Torres y Luis Casillas que aunque ya no están con nosotros nos dejan su alegría y su magnífica visión del mundo. Al coordinador de posgraduación del IMECC, el Profesor Aurelio Casillas, que me apoyó en todo momento. A los Profesores Von Zuben, João Florindo, Estevão Esmi, Humberto Sossa, Manuel Graña y Gerhard Ritter que contribuyeron con sus comentarios en las bancas examinadoras. Especialmente agradezco al Profesor Peter Sussner que me destinó su tiempo y me orientó con compromiso y dedicación.

Agradezco al Consejo Nacional de Ciencia y Tecnología de México (CONACYT) por haber financiado este proyecto (CVU 367565). El apoyo institucional es indispensable para el progreso de la investigación científica.

Resumo

Há um interesse muito alto nos métodos de aprendizagem profunda (DL), eles ganharam um amplo reconhecimento na pesquisa experimental e têm mostrado excelentes resultados em um grande número de aplicações. Particularmente, cabe destaque às arquiteturas de DL desenvolvidas para executar a tarefa de classificação envolvendo métodos de otimização para treinamento, como os algoritmos de descida estocástica do gradiente (SGD) e backpropagation (BP). Os perceptrons morfológicos (MPs) podem ser definidos como redes neurais morfológicas feedforward (MNNs) projetadas para realizar classificação. Um neurônio de uma MNN usa uma operação de morfologia matemática (MM) para agregar entradas neurais. Como as operações de MM geralmente não são diferenciáveis e o SGD usa derivadas parciais (gradientes) para calcular as direções nas quais os parâmetros são atualizados, algumas modificações para treinar MPs via SGD foram desenvolvidas por alguns autores. As modificações visam calcular quais direções os parâmetros devem seguir para obter melhores superfícies de decisão. Nesse sentido, duas estratégias envolvidas nessa modificação são: o uso de funções de pulso suavizado e a inserção de valores zero (sem direção) onde as derivadas parciais não existem. Embora existam alguns modelos híbridos/morfológicos que fazem uso de algoritmos SGD modificados, houve menos estudos anteriores procurando por uma perspectiva híbrida de DL/morfologia. Além disso, até onde sabemos, a hibridização entre MP e DL não foi uma questão central de nenhum estudo anterior. Os objetivos desta tese foram desenvolver uma maneira de descrever uma nova arquitetura híbrida DL/ morfológica envolvendo camadas tradicionais de DL e camadas morfológicas; e comparar experimentalmente os resultados subsequentes com outros classificadores existentes. Para esse fim, propomos uma extensão da estrutura dos MPs para anéis arquimedianos totalmente ordenados. Analisamos operações elementares de anéis arquimedianos totalmente ordenados em termos de MPs e estudamos essas operações em termos de ideias geométricas das arquiteturas de MP. Dois operadores, os descritores \mathbb{I} -set e os agregadores \mathbb{L} - fuzzy foram introduzidos. No processo, descobrimos uma unidade de computação diferenciável que pode ser usada para executar uma estratégia semelhante aos MPs tradicionais. Os resultados imediatos foram as relações entre as unidades de computação dos MPs e nossa unidade de computação proposta em relação às distâncias L_∞ e L_2 , respectivamente. Isso nos levou ao desenvolvimento de uma arquitetura geral baseada em reticulados que pode ser treinada pela SGD e, além disso, levou ao desenvolvimento de camadas morfológicas, nas quais as funções *forward pass* e *backward pass* de BP podem ser implementadas. Aqui, três classes de modelos foram propostas: uma rede morfológica feedforward puramente baseada, na qual a nova camada morfológica é usada na fase final; e dois modelos feedforward híbridos DL/morfológicos. O primeiro modelo híbrido utiliza camadas lineares totalmente conectadas com ativação não linear como parte do DL. O outro utiliza camadas convolucionais com métodos de ativação não linear, downsampling,

expansão de dados e regularização como parte do DL. Ambos os modelos híbridos utilizam camadas morfológicas para realizar a classificação. Consequentemente, o modelo híbrido, no qual a convolução está envolvida, é capaz de extrair características de estruturas espaciais, como imagens. Além disso, a fim de melhorar os resultados, uma revisão dos algoritmos SGD e técnicas conhecidas para melhorar os resultados da BP, como momento, taxa de aprendizado, funções de perda e mini-lotes, foi revisada. O conjunto de modelos híbridos/morfológicos foi validado experimentalmente. Os resultados e comparações de desempenho com outros classificadores sugerem uma capacidade de generalização de alto nível e algumas vantagens dessas novas abordagens. Esta tese oferece uma visão sobre novas classes de técnicas baseadas em reticulados que podem ser combinadas com modelos clássicos baseados em BP. Concluímos que a perspectiva da morfologia matemática pode desempenhar um papel importante no desenvolvimento de novas técnicas de DL.

Palavras chave— aprendizado profundo, redes neurais convolucionais, morfologia matemática, aprendizagem supervisionada

Abstract

There is a very high interest in deep learning (DL) methods and they have gained a wide recognition as experimental research, and they have shown excellent results in a large number of applications. Particularly, DL architectures were designed to perform classification involving optimization methods for training, such as stochastic gradient descent (SGD) and backpropagation (BP) algorithms. Morphological perceptrons (MPs) can be defined as feedforward morphological neural networks (MNNs) designed to perform classification. A neuron of an MNN uses an operation of mathematical morphology (MM) in order to aggregate the neuronal input. Since operations of MM are usually not differentiable and SGD uses partial derivatives (gradients) to compute the directions in which the parameters are updated, some modifications to train MPs via SGD have been developed by some authors. The modifications are intended to compute which directions for parameter adjustment should be followed to achieve better decision surfaces. In this sense, two strategies involved in these modification are: the use of smoothed pulse functions and the insertion of zero values (no direction) whenever the partial derivatives do not exist. Although there exist some morphological/linear models that make use of modified SGD algorithms, there have been less previous studies from a hybrid morphological/linear based on a DL perspective. Moreover, to our knowledge, hybridization between MP and DL has not been a central issue of any previous study. The aims of this thesis were to develop a way to describe a novel DL/morphological hybrid architecture involving traditional DL layers and morphological layers; and to experimentally compare the subsequent results with other existent classifiers. To this end, we propose an extension of MPs framework to archimedean totally ordered rings. We analyzed elementary operations of archimedean totally ordered rings in terms of MPs and we studied these operations in terms of the geometrical ideas of MP architectures. Two operators, \mathbb{L} -set descriptors and \mathbb{L} -fuzzy aggregators were introduced. In the process, we discovered a differentiable unit of computation that can be used to perform a similar strategy of traditional MPs. Immediate results were the relations between MPs units of computation and our proposed unit of computation with respect to distances L_∞ and L_2 , respectively. Those relations led us to the development of a general lattice based architecture that can be trained by SGD and furthermore, it led to the development of morphological layers, in which forward and backward functions of BP can be implemented. Here, three classes of models were proposed: a purely based morphological feed forward network, in which the new morphological layer is used at the final stage; and two hybrid feed forward DL/morphological models. The first hybrid model makes use of linear fully connected layers with non-linear activation as the DL part. The other makes use of convolutional layers with non-linear activation, downsampling, data expansion and regularization methods as the DL part. Both hybrid models make use of morphological layers to perform classification. Consequently, the hybrid model, in which convolution is

involved, is able to extract features from spatial structures, such as images. Moreover, in order to improve the results, a review on SGD algorithms and well known techniques to improve the results of BP such as momentum, learning factor, loss functions, mini-batches were reviewed. The set of hybrid/morphological models was experimentally validated. The results and comparisons of performance with other state of the art classifiers suggest a high-level capacity of generalization and some advantages of these novel approaches. This thesis offers an insight into new classes of lattice based techniques that can be hybridized with BP based models. We conclude that the mathematical morphology perspective can play an important role in the development of new DL techniques.

Keywords— deep learning, convolutional neural networks, mathematical morphology, supervised learning

List of Figures

Figure 1 – The Lattice of Positive Integer Pairs Ordered Component-wise.	26
Figure 2 – Generated Sets by Aggregation-Descriptor Pairs	46
Figure 3 – Confusion Matrix	58
Figure 4 – Contour Plots.	60
Figure 5 – Oscillations of SGD.	64
Figure 6 – Multilayer Perceptrons.	67
Figure 7 – Paths between Layers.	68
Figure 8 – Backpropagation Flow Diagram.	71
Figure 9 – Example of Average Pooling	74
Figure 10 – Example of Max Pooling	75
Figure 11 – Example of Dropout in FC layers	77
Figure 12 – Example of Dropout	78
Figure 13 – Example of Dropout Connect	78
Figure 14 – Autoencoders	82
Figure 15 – Basic CNN architecture	83
Figure 16 – Example of CNN Processing - First layers.	84
Figure 17 – Example of CNN Processing - Max-pooling layer.	84
Figure 18 – Example of CNN Processing - Second Convolution and Average Layer.	85
Figure 19 – MP/CL s-th class module.	92
Figure 20 – Morphological Perceptron with Dendritic Structure	94
Figure 21 – Elimination and Merge Principles	96
Figure 22 – MP/CL’s Decision Surface Example	99
Figure 23 – HMLP Topology	107
Figure 24 – Generalized Morphological Perceptron	111
Figure 25 – Spiral and Included Datasets	117
Figure 26 – FC-SD-Softmax Model Test on Spiral and Included Datasets	118
Figure 27 – Accuracies on MNIST dataset	131
Figure 28 – Conv-SD Bar-chart Common Errors for MNIST.	132
Figure 29 – MNIST Confusion Matrices	132
Figure 30 – Set of Misclassified Images	133
Figure 31 – Conv-FC Bar-chart Common Errors for the Models in the MNIST Experiment	134
Figure 32 – Results of Committees for Classification	134
Figure 33 – Design of the VBSL-UPV experiment	135
Figure 34 – ROBOTICS Analysis of the Results	139
Figure 35 – Examples of CIFAR-10 Images.	140

Figure 36 – CIFAR-10 Confusion Matrices 142

List of Tables

Table 1 – Aggregation-Descriptors Pairs	51
Table 2 – Activation Functions	76
Table 3 – Reduced Notations to Describe Layers.	82
Table 4 – Algorithms Properties	108
Table 5 – Hyperparameters computed for the datasets selected from UCI.	122
Table 6 – Classification Accuracies and F1-Scores in the Testing Stage.	126
Table 7 – Accuracy and Model’s Configuration for MNIST-1000 Digits Dataset	129
Table 8 – MNIST Digits Dataset Results	135
Table 9 – Number of Observations in the VBSL-UPV dataset.	136
Table 10 – Computed Hyperparameters for VSLR-UPV dataset.	137
Table 11 – Results on the VBSL-UPV dataset.	138
Table 12 – Results on the Dataset CIFAR-10 (Error rate %).	139
Table 13 – Configuration of the Conv-SD models for CIFAR-10	141

List of Algorithms

Algorithm 1 – Training Algorithm MP/CL	98
Algorithm 2 – Divide and Conquer Training Algorithm	100
Algorithm 3 – Differential Evolution Training Algorithm	101
Algorithm 4 – Ellipsoid Training Algorithm	103

Contents

1	Introduction	16
2	Some Relevant Concepts of Lattice Theory	22
2.1	Historical Context	22
2.2	Posets and Lattices	24
2.3	Notions on Lattice Ordered Groups and Rings	31
2.4	Bounded Extensions of ℓ -groups and ℓ -rings	37
2.5	Mathematical Morphology on Complete Lattices	39
2.6	Aggregations and \mathbb{I} -set Descriptors on Archimedean σ -Rings	42
3	Notions on Classification using Backpropagation and Deep Learning	52
3.1	Supervised Learning in Classification Problems	55
3.2	Introduction to Gradient Descent	58
3.3	Deep Learning and Backpropagation	65
3.4	Some Deep Learning Layers	70
3.5	Some Deep Learning Architectures	80
4	A Review of Morphological Perceptrons and Related Models	86
4.1	The Original Morphological Perceptron Model	88
4.2	Morphological Perceptron with Competitive Learning	91
4.3	Dendrite Morphological Neural Networks	92
4.4	MP and DMNN Training Algorithms	96
4.5	Hybrid Morphological Models	103
4.6	Challenges and Issues of Morphological Perceptrons	108
5	Generalized Morphological Perceptrons	110
5.1	General Description of a GMP Model	111
5.2	A Review on Aggregator and Descriptor Combinations	112
5.3	Generalized Morphological Perceptron in Deep Learning	113
5.4	The Sum-Dot Model	114
5.5	FC-SD Models	116
5.6	Conv-SD Models	118
6	Some Experimental Results	120
6.1	General Classification Problem Datasets	120
6.2	MNIST1000 Digits Dataset	128
6.3	MNIST Digits Dataset	130
6.4	Visual Self-Localization Dataset	134
6.5	CIFAR-10 Dataset	139
7	Conclusions	143

BIBLIOGRAPHY 145

1 Introduction

Generally speaking, MM is a theory related to set, lattice and algebra concepts, and has a large number of applications in signal processing. In fact, MM was developed as the PhD Thesis of Jean Serra under the supervision of Georges Matheron, who was working on analysis of minerals (SERRA; VERCHERY, 1973). One of the results led to the development of a device, named "Texture Analyser" that computes the so-called binary hit-or-miss transformation (KLEIN; SERRA, 1972). The formal description of the operations performed by this device were derived from the concepts of erosion and dilation first studied in (MATHERON, 1975), resulting in many interesting operators for image processing (MATHERON; SERRA, 2002). Then, binary images were the first focus of MM until the middle of the 1970s (SERRA, 1983). Later, some approaches to extend the study from binary to grayscale structures were developed (STERNBERG, 1986). For this study, it is of interest to review the umbra approach to grayscale MM, as discussed in Chapter 4. The subsequent algebraic generalization to complete lattices, which is the most accepted framework of MM, is a fundamental piece of many MM applications (RONSE, 1990).

In the 1990s, besides a wider recognition and successful applications in the field of image analysis, MM was adopted in the field of machine learning, especially in Morphological Neural Networks (MNNs) which are intimately related to Artificial Neural Networks (ANNs) and are an important inspiration of this work. Some examples of applications are: land mine detection (GADER; KELLER; NELSON, 2001); Dilation erosion linear perceptrons (DELP) (ARAÚJO; OLIVEIRA; MEIRA, 2012) in time series prediction; Morphological associative memories (MAMS) (RITTER; SUSSNER; LEON, 1998) in classification, compression and hyperspectral image analysis (SUSSNER; VALLE, 2006b; VALDIVIEZO-NAVARRO; URCID-SERRANO, 2007; SCHMALZ; RITTER, 2006; GRAÑA et al., 2003); Fuzzy morphological associative memories (FMAMs) and intervalar FMAMs (IV-FMAMs) in time series prediction, non-linear systems identification, image classification, among others (SUSSNER; VALLE, 2006b; SUSSNER; VALLE, 2006a; VALLE; SUSSNER, 2008b; ESMI et al., 2014; SUSSNER; SCHUSTER, 2018; SUSSNER; ESMI, 2019).

Broadly speaking, ANNs are connectionist systems vaguely inspired by biological neurons. ANNs units of computation are known as artificial neuron. ANN models abstract some elementary behaviours and operations that a biological neuron performs. A single artificial neuron receives signals, then it processes the signal and produces an output that travels to other units, similar to the axon signal flow process. So, every artificial neuron receives a set of input signals that represents excitatory and inhibitory postsynaptic potentials at neural dendrites and each connection between units resembles the synapses in a simplified way. The output of each neuron is computed by some aggregation function followed by an

activation function (usually nonlinear). Applications of ANNs are widely spreading and permeating the computer sciences, particularly, machine learning. In particular, MNNs are a type of artificial neural network that performs an operation of mathematical morphology at each node. As a generality, their nodes, morphological neurons, apply morphological operators before some activation function. According to Heijmans, any attempt to define morphological operators would be either very restrictive and will exclude operators, or, too general, leading to a theory of everything (HEIJMANS, 1994). Nevertheless, four definitions of elementary operators of mathematical morphology (MM) can be defined according to the algebraic framework of MM: dilation, erosion, anti-dilation, and anti-erosion. Specifically, (BANON; BARRERA, 1993) proved that all mappings from one complete lattice to another can be established by combining those four fundamental operations. As aforementioned, MNNs have been applied with great success in many problems of machine learning.

Machine learning (ML) studies algorithmic models used to perform specific tasks without the use of explicit instructions. Some of the most usual tasks studied in ML are: unsupervised, supervised, and feature learning, anomaly detection, rule based association, among many others. The differences between tasks studied in ML are: the type and manner in which input and output are given, the type of external rewards and which behavior is expected from the model. It should be noted that ANNs are present in almost the entire spectrum of ML and all tasks are usually connected. For instance, supervised and unsupervised learning can be improved by feature learning, selecting better and more representative characteristics from the initial data. Inside an artificial neuron, the weights and inputs of the synapses interact and are aggregated into a single value. The way a neuron gathers the input from other previous neurons is called aggregation function. Once an artificial neuron receives and aggregates the inputs from other neurons into a single value, an activation function performs the computation of the final output. A prominent example of aggregation function is the inner product drawn from linear algebra of the synaptic weight vector and the input vector. Linear algebra is a subalgebra of image algebra (RITTER; DAVIDSON; WILSON, 1987; DAVIDSON; SUN, 1991) and can be used to formulate traditional ANN models.

A rigorous theoretical basis for MNNs introduced in (RITTER; SUSSNER, 1996) enabled the possibility of transforming the traditional neural networks into morphological counterparts and demonstrated the capability of MNNs to solve conventional computational problems, for example: boolean logical gates, class separation and pattern reconstruction. Binary morphological associative memories (MAMs) were studied as a morphological counterpart of Hopfield networks. Furthermore, (SUSSNER, 1998) introduced the basis of morphological perceptron (MP) and the first training algorithm, specifically the single layer morphological perceptron (SLMP) analogous to the architecture of single layer Rosenblatt perceptrons. This second approach is one of the main issues discussed in this work.

Some of the first efforts in research on MPs were to unify MPs, multilayer feed-forward networks, adaptative filters and rank/linear nodes into one framework (PESSOA; MARAGOS, 1998). Modular neural networks and hybrid morphological/rank linear models (MRI) were introduced and applied successfully (PESSOA; MARAGOS, 1998; AUDA; KAMEL, 1999; PESSOA, 1999). (PESSOA; MARAGOS, 2000) introduced a modified backpropagation optimization algorithm using a combination of typical perceptrons (sum of product function) and MRL filters, achieving outstanding results in the handwritten classification digits problem. This combination of neurons led to models that results in more complex decision boundaries than the hyperboxes presented in previous works.

A plausible biological inspiration of MPs was studied in (RITTER; URCID, 2003; RITTER; IANCU; URCID, 2003). Dendrite morphological neural networks (DMMNs) emerged as an improvement of MNNs that incorporate a more realistic model of single neuron computation in artificial neural networks (RITTER; URCID, 2003). In their work, Ritter et al., tied the dendrite process of a neuron to the morphological operations executed by the MNNs. Therefore, MP models were formally enhanced with a theoretical description in terms of dendrites.

In (RITTER; IANCU, 2003; RITTER; IANCU; URCID, 2003) two algorithms to train single layer dendritic morphological perceptrons for supervised learning were discussed. Furthermore, a theoretical result on the capability of these models to separate compact subsets collections was demonstrated. DMNNs applications to classification can be found in (RITTER; URCID, 2007; SOSSA; GUEVARA, 2014; ZAMORA; SOSSA, 2017).

An important step forward was introduced in the second half of the 2000s in (SUSSNER; ESMI, 2009b) who incorporated a winner take all output layer to the MP architecture. The resulting model is called morphological perceptron with competitive layer (MP/CL). Furthermore, they provided a more comprehensive lattice theoretical background for MPs and developed a multiclass classification algorithm for MP/CL training, which does not depend on the sequence in which the training data are presented to the network and generates the number of morphological neurons in an automatic fashion. In addition, a proof of the convergence in a finite number of steps was given in (SUSSNER; ESMI, 2009a; SUSSNER; ESMI, 2009b).

Besides MNNs, a major topic to be investigated in this thesis is deep learning (DL), both inspired in ideas of ANN. We are especially interested in hybrid methods that make use of both morphological and deep learning notions. Broadly speaking, a deep learning architecture uses a cascade of layers with nonlinear processing units to extract information through the transformation of variables. Each layer uses the output of the previous layer as input, and each processing unit performs a multivariable operation. The algorithms can use supervised learning or unsupervised learning, and applications include data modeling and pattern recognition. The goal is to learn more complex levels of representation that correspond to different degrees of abstraction. In other words, a layer receives an abstract

representation learned by the previous layer and learns a new, higher-level representation, forming a hierarchy of concepts. There is no single standard for the number of layers that makes a neural network model deep. However, much of the literature considers that deep learning involves at least two intermediate transformations (layers). In this thesis we are focused on deep learning algorithms related with backpropagation (BP), in which the final task is classification. In this way, a good example of deep learning architecture used to perform classification are those MPL-BP with more than two hidden layer. Nevertheless, other deep learning architectures such as convolutional Neural Networks (CNN) (LECUN; BENGIO et al., 1995) and auto-encoders (GOODFELLOW; BENGIO; COURVILLE, 2016) fit the description of our interest.

Whereas DL has been strongly linked to gradient optimization techniques such as stochastic gradient descent (SGD), BP and multilayered architectures, most morphological perceptron algorithms are strongly concerned with constructive methods to combine morphological neurons to enclose training patterns in bounding boxes, hyperplanes, parallel hyperboxes and rotated ellipses to generate nonlinear decision surfaces. Specifically, in contrast to ANN models proposed in (ROSENBLATT, 1961), most of the learning rules and algorithms to train MPs build a set of hyperboxes by means of incremental processes (SUSSNER; ESMI, 2009a). Since classical MP models are described by minimax algebra and minimax operations, which are not differentiable, approaches purely based on gradient descent are usually out of the question. Hence, Rosenblatt's Perceptrons and MP models have taken forked roads. A new approach is therefore needed for building a DL architecture with a morphological part.

Some algorithms compute hyperboxes using approximation methods. In short, (ZAMORA; SOSSA, 2017) show how the gradient descent principle can be applied to improve the results computed by other constructive algorithms. Recently, (ARCE et al., 2017) replaced hyperboxes by hyper-ellipses rotated by covariance matrices and presented a constructive training procedure that relies on the inclusion nature of MP models. However, morphological learning rules purely based on SGD and related to DL have rarely been studied directly.

An alternative, to make use of approximation methods such as SGD in the morphological context, is to consider hybrid methods, in which a part of the whole net has morphological nodes and the nodes of other parts compute some other non-linear function. For instance, hybrid morphological linear perceptrons (HMLP), trained by extreme learning machine, employ a random weights initialization of a layer composed of regularized linear nodes and morphological ones (SUSSNER; CAMPIOTTI, 2020). Other methods, such as dilation erosion linear perceptron (DELP) and MRI, make use of pulse functions to compute which direction the set of parameters must follow to improve the output. In Chapter 4 we discuss some hybrid morphological models and the ideas behind them.

The aim of this thesis is to develop a more sophisticated method for classification that

combines the deep learning with stochastic gradient and the MNN approaches, using morphological operators in at least one layer. Although some researchers have developed some morphological algorithms that make use of gradient descent algorithm (PESSOA; MARAGOS, 1996; PESSOA, 1999; ARAÚJO; OLIVEIRA; MEIRA, 2012; ZAMORA; SOSSA, 2017; HERNÁNDEZ; ZAMORA; SOSSA, 2018; SUSSNER; CAMPIOTTI, 2020), no study, to our knowledge, has considered a DL method in which at least one layer has morphological nodes. Especially when a gradient descent algorithm is used in DL, morphological nodes are not considered due the differentiability problems of the operators based on morphology. One way to overcome the non-differentiability problem of traditional MNN models is to extend their framework. We are interested, especially in archimedean ℓ -rings that were first introduced in (BIRKHOFF, 1987). In Chapter 2 these concepts are reviewed and discussed.

In order to achieve this extension, we first review the morphological perceptron architecture and its variants. Furthermore, we capture the geometrical nature of its elementary operations and then, we describe a framework, in which two operators play an essential role: \mathbb{I} -set descriptors and \mathbb{L} -fuzzy aggregators, described in terms of the lattice framework. The main idea is to analyze the behavior of all elementary operators in totally ordered archimedean ℓ -rings and to fit the geometrical ideas and classification rules of MPs. The first insight is the fact that to every archimedean ℓ -ring an archimedean ℓ -group is attached. These ℓ -groups can be extended to bounded lattice ordered groups in which minimax algebra is defined. Therefore, most of the mathematical morphological models studied before can be described in terms of ℓ -rings. As far as we know, no previous research has considered archimedean totally ordered ℓ -rings as the framework of MPs.

Specifically, we examined the structure of MPs by means of \mathbb{I} -set descriptors and \mathbb{L} -fuzzy aggregators and we present some novel results. We also analyze the dot operator introduced in ℓ -rings as \mathbb{I} -set descriptor. As a result, we describe a differentiable operator that fits the geometrical and the learning rule ideas applied in MPs. Moreover, we developed an architecture trainable by BP that comprises morphological nodes. The resulting architecture was named generalized morphological perceptron (GMP). Moreover, once we have an MM based differentiable unit of computation, backward and forward functions of a morphological layer is described based in the aforementioned operators of Sum-Dot in archimedean totally ordered ℓ -rings.

In addition, hybrid models with GMP layers based on elementary ℓ -rings operators are fully described. In contrast to (ZAMORA; SOSSA, 2017), we present a morphological architecture that learns by SGD without any constructive method involved. A second architecture that involves one or more traditional fully connected layers followed by an activation function and layer based in ℓ -rings with a softmax-logit loss function is also explored. Unlike (HERNÁNDEZ; ZAMORA; SOSSA, 2018), in the last aforementioned architecture the morphological layer performs a backpass function to previous layers.

Furthermore, a set of simulations were performed and a comparison among other lattice and morphological methods for supervised learning was conducted. Moreover, a larger comparison with state of the art methods for supervised learning is presented.

Therefore, we organized this text as follows:

Chapter 2 contains the theoretical concepts used throughout this thesis. In Sections 2.2 and 2.3, we review posets and lattices. Sections 2.4 and 2.5 contain reviews on bounded ordered posets and an introduction to MM from the algebraic perspective. Last, in Section 2.6 a brief and intuitive discussion of the results is presented.

In Chapter 3, a brief review on machine learning is provided, with special emphasis on multiclass classification, how SGD is applied and some DL theory related with BP. In Section 3.2, some techniques that improve SGD such as learning step, momentum and regularization are discussed. Sections 3.3 and 3.4 contain an introduction to DL and how SGD is applied through BP. Besides, a set of existent layers relevant to us is listed. This chapter concludes with some self-explanatory feed forward deep neural networks examples trained by SGD, such as: autoencoders, convolutional neural networks.

Chapter 4 presents an introduction to MNNs, and specially MPs. Special attention to morphological/hybrid architectures and its respective algorithms to train is given. Sections 4.1, 4.2 and 4.3 are focused on the historical and algebraic foundations of MPs. Last, in Sections 4.4 and 4.5, we review some of the MP and hybrid architectures previously described in literature.

Chapter 5 introduces the generalized morphological perceptron. Also, an alternative formulation of MPs units of computation, in terms of \mathbb{L} -set descriptors and \mathbb{L} -fuzzy aggregators, is presented. Moreover, we discuss the relation between this alternative formulation and the results presented in Section 2.6. In Section 5.3, we present the architecture of the generalized morphological perceptron. In particular, we present two layers, one layer that computes an \mathbb{L} -set descriptors and a \mathbb{L} -fuzzy aggregator at each unit of computation, and the other one that computes a union or a intersection of nodes by class. Sections 5.4, 5.5 and 5.6 introduce three hybrid models that make use of the aforementioned layers.

Experiments in classification using several datasets are presented in Chapter 6. The performance of our proposed architectures are compared with some other algorithms from the literature, in particular those based on lattice theory and some competitive DL classifiers. Finally, conclusions and final remarks can be found in Chapter 7.

2 Some Relevant Concepts of Lattice Theory

I took a course in lattice theory from Oystein Ore while a graduate student at Yale in the fall of 1954. The lectures were scheduled at 8 a.m., and only one other student attended besides me, María Wonenburger. It is the only course I have ever attended that met at 8 o'clock in the morning. The first lecture was somewhat of a letdown, beginning with the words: "I think lattice theory is played out".

Gian-Carlo Rota, *The Many Lives of Lattice Theory*.

2.1 Historical Context

It is hard to follow the historical development of the idea behind lattices, particularly before the 19th century. Nevertheless, the emergence and its formalization was not in the middle of nowhere and some mathematicians worked together lattice structures in the middle of nineteenth century, although their results did not have a great response. The first lattice structure appears in Boolean algebras ([BOOLE](#); [GRATTAN-GUINNESS](#); [BORNET](#), 1999). Although the term lattice was not available, the interest in algebraic logic was a paramount issue to the development and the future formalization of lattice theory. Three mathematicians stand out in the contribution to studies of the algebraic logic: G.Boole studied the logic as algebra, Ch. S. Peirce contributed to the axiomatization and improvement of Boole's calculus and, E. Schröder who proved the independence of the distributive law in Boole's calculus contrary to Peirce's proposal ([PEIRCE](#), 1880). This generalization led Schröder to study two systems of algebraic logic with different lattice structures which did not satisfy distributivity, the first non-distributive lattices. As a result Schröder started to study two systems of algebraic logic, and therefore he distinguished two lattice structures: "identity calculus" as the specialized Boolean algebra and "logical calculus with groups" as a more general system which did not satisfy the distributive law

([SCHRÖDER, 1890](#)).

Despite these advances, these abstract algebraic structures did not attract much attention during the last decades of the nineteenth century. However, the mathematician Dedekind established the foundations of number theory, an area that was fundamental for the development of lattice theory.

After a long recess, Karl Menger introduced the axioms of projective geometries which turned out to be modular lattices. Fritz Klein-Barmen introduced the German term Verband ([MEHRTENS, 1979](#)) for lattice, and later, concentrated on an axiomatic formulation of the theory. After, Garret Birkhoff, inspired by Hasse diagrams, introduced the word "lattice" for the first time. Also, Birkhoff formulated lattice theory as an essential tool for algebra in his book, Lattice Theory ([BIRKHOFF, 1940](#)) published in 1940.

However the study of abstract structures was not in fashion in the second half of the nineteenth century, and therefore the systems of Schröder and Dedekind research did not stimulate further investigation. It was not until the 1930's that a research of this kind obtained a wider response. In 1920, Karl Menger presented the set of axioms characterizing projective geometries which are in fact complemented modular lattices. The biggest merits in the early developments of lattice theory belong to Birkhoff who also conceived lattice as a basic tool in algebra. In his early articles he rediscovered, among others, Schröder and Dedekind's results. The book Lattice Theory, published in 1940, attracted interest from almost all areas of the mathematics and was the beginning of a proliferating list of applications of the so-called "young brother of group theory".

An extended edition of the book Lattice Theory was published in 1948 including the foundations of the next decades, which became a reference for modern algebra. Nevertheless, the great expectation in this new area diminished in the 1950s. Rota's historical review of lattice theory claims: "Never in the history of mathematics has a mathematical theory been the object of such vociferous vituperations as lattice theory. Dedekind, Jónsson, Kurosh, Malcev, Ore, von Neumann, Tarski, and most prominently Garrett Birkhoff have contributed to a new vision of mathematics, a vision that has been cursed by a conjunction of misunderstanding, resentment, and raw prejudice" ([ROTA, 1997](#)).

In 1967 a third edition of the book "Lattice theory" ([BIRKHOFF, 1967](#)) refreshed the subject, gathered the advances of the theory and launched the new challenges that appeared until then. In this edition, Birkhoff points out that lattice theory had not had the attention that it deserves,

The General Lattice Theory book published by Grätzer ([GRÄTZER, 2002](#)) in 1978 left clear the transformation that mathematics underwent with the development of lattice theory. From that time on, many of the advances made in the last decades in lattice theory became specialized areas of study. Furthermore, lattice theory has been a growing and fertile field of research in terms of both its applications and its theoretical framework.

2.2 Posets and Lattices.

The conventional way to introduce the theory of lattice is to start from the partially order relation which generalizes the concept of ordering or sequencing the elements of a set. A more general concept is the pre-order concept from order theory, pre-orders satisfy reflexivity and transitivity. Two specialized branches are defined as a symmetric pre-order or equivalence relation and the anti-symmetric pre-order or partially ordered set which is of our interest. To dive into lattice theory we will start with the definition of partially ordered set.

Definition 1 (Partially ordered set). *A pair $(\mathbb{P}, \leq_{\mathbb{P}})$ where \mathbb{P} is a set and $\leq_{\mathbb{P}}$ is a binary relation, is named partially ordered set (poset) if and only if for any $x, y, z \in \mathbb{P}$ the pair satisfies the following three conditions:*

1. $x \leq_{\mathbb{P}} x$ (**reflexivity**).
2. $x \leq_{\mathbb{P}} y$ and $y \leq_{\mathbb{P}} x$ if and only if $x = y$ (**antisymmetry**).
3. $x \leq_{\mathbb{P}} y$ and $y \leq_{\mathbb{P}} z$ if and only if $x \leq_{\mathbb{P}} z$ (**transitivity**),

The notation $\leq_{\mathbb{P}}$ is used to avoid possible missinterpretation when there are other binary relationships in the same context.

Two common examples of posets are: a set of n humans together with the binary relation given by the genealogical descendancy; and a set of words ordered by the alphabetic relation. In words, lattice theory is about the study of the binary relations read "is less or equal", "is contained in", "is part of" that arises from the aforementioned conditions.

Every partially ordered set \mathbb{P} gives rise to an opposite or **dual partially ordered set** which is often denoted by \mathbb{P}^* . This dual order is defined in the same set, but with the inverse order, that is: $x \leq_{\mathbb{P}^*} y$ holds in \mathbb{P}^* if and only if $y \leq_{\mathbb{P}} x$ holds in \mathbb{P} or equivalently $x \geq_{\mathbb{P}} y$.

Definition 2 (Lower and upper bounded sets). *Given a poset (\mathbb{P}, \leq) , for any $M \subseteq \mathbb{P}$, the **upper bound** and the **lower bound sets** are given by $\mathcal{U}(M) = \{u \in \mathbb{P} : x \leq u, \forall x \in M\}$ and $\mathcal{V}(M) = \{v \in \mathbb{P} : x \geq v, \forall x \in M\}$, respectively.*

Informally speaking, an upper bound of the subset $M \subseteq \mathbb{P}$ is an element of P "greater than or equal to" (\geq) every element of the subset M . A dual statement can be done for a lower bound using the relation "less than or equal to" (\leq). Especial cases of upper and lower bounds are the supremum and the infimum defined below.

Definition 3 (Supremum and infimum). *The **supremum** or least upper bound (l.u.b) of a subset $M \subseteq \mathbb{P}$ is denoted $(\bigvee M)$ and satisfies $x \in \mathcal{U}(M)$ and $x \leq y, \forall y \in \mathcal{U}(M)$.*

Analogously, $x \in \mathcal{U}(M)$ is called the **infimum** or greatest lower bound (g.l.b.) and denoted $(\bigwedge M)$ if satisfies $x \in \mathcal{V}(M)$ and $x \geq y, \forall y \in \mathcal{V}(M)$.

It is easy to verify that if there exists a supremum (dually an infimum) then it is unique. The supremum (join) and the infimum (meet) of a pair of elements is denoted by \vee and \wedge , respectively. For $M \subseteq P$ we denote the supremum and infimum using the symbols $\bigvee M$ and $\bigwedge M$, whenever they exist. In addition, we say that M is bonded if $\bigvee M$ and $\bigwedge M$ exists.

Through these basic definitions is possible to prove the anti-circularity of a poset, that is:

Lemma 1. *Given a poset (\mathbb{P}, \leq) and $x_1, \dots, x_n \in \mathbb{P}$, if $x_1 \leq x_2 \leq \dots \leq x_n \leq x_1$ then $x_1 = x_2 = \dots = x_n$.*

As we have mentioned before, a lattice is an abstract structure widely studied in order theory and abstract algebra. A lattice is a partially ordered set in which every pair of elements has a supremum and an infimum.

Definition 4 (Lattice). *A poset (\mathbb{L}, \leq) is called **lattice** if and only if $\forall x, y$ there exist $x \vee y = \bigvee x, y$ and $x \wedge y = \bigwedge x, y$ in \mathbb{L} .*

A lattice (\mathbb{L}, \leq) is called **bounded** if and only if \mathbb{L} has a least element $\bigwedge \mathbb{L}$ and a greatest element $\bigvee \mathbb{L}$. The least $\bigwedge \mathbb{L}$ and the greatest $\bigvee \mathbb{L}$ elements of a bounded lattice \mathbb{L} are denoted by $0_{\mathbb{L}}$ and $1_{\mathbb{L}}$.

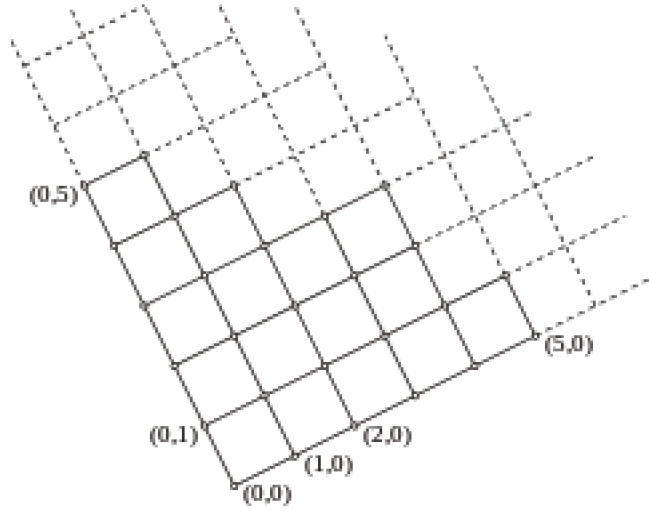
A lattice (\mathbb{L}, \leq) is called **complete** if and only if for any $M \subseteq \mathbb{L}$ there exist $\bigvee M$ and $\bigwedge M$ in \mathbb{L} . Obviously, every complete lattice is bounded and every lattice with a finite number of elements is complete. Furthermore, a lattice (\mathbb{L}, \leq) is called **totally ordered** or chain if $x \leq y$ or $y \leq x \forall x, y \in \mathbb{L}$.

The Cartesian square of the natural numbers, ordered so that $(a, b) \leq (c, d)$ if $a \leq c$ and $b \leq d$ is a common example of a lattice. The pair $(0, 0)$ is the bottom element and there is no top. In this case, the lattice is not upper bounded and it is not complete. Figure 1 shows a diagram of this example.

Definition 5 (Conditionally complete lattice). *A lattice (\mathbb{L}, \leq) is said to be **conditionally complete** if every non-empty bounded subset $S \subset \mathbb{L}$ has a supremum and an infimum in \mathbb{L} .*

A subset $\mathbb{M} \subseteq \mathbb{L}$ is a **meet-sublattice** of \mathbb{L} if the infimum operation of a lattice \mathbb{L} is defined for any two $x, y \in \mathbb{M}$ and is called **join-sublattice** if the supremum of a lattice \mathbb{L} is defined for any pair of elements of \mathbb{M} . A subset \mathbb{M} of \mathbb{L} that is a join and a meet-sublattice is called sublattice of \mathbb{L} . In other words, a subset $\mathbb{M} \subseteq \mathbb{L}$ is a **sublattice** of \mathbb{L} if \mathbb{M} is a lattice and shares infimum and supremum operations with \mathbb{L} .

Figure 1 – The Lattice of Positive Integer Pairs Ordered Component-wise.



Given $x \leq y$ in a lattice \mathbb{L} , the closed interval $[x, y] = \{z \in \mathbb{L} : x \leq z \leq y\}$ is a sublattice of \mathbb{L} , formally:

Definition 6 (Convex sublattice). A subset \mathbb{M} of a lattice \mathbb{L} is a **convex sublattice** of \mathbb{L} , when $x, y \in \mathbb{M}$ implies that every $z \in \mathbb{L}$ such that $x \wedge y \leq z \leq x \vee y$, is an element of \mathbb{M} .

Many important lattices that are not complete lattices have the conditionally complete property. For example, \mathbb{R} (ordered the usual order relation) is conditionally complete. The following statements are some results on conditionally complete lattices.

Theorem 1. For a lattice \mathbb{L} to be conditionally complete, it is sufficient that every bounded non-empty subset of \mathbb{L} has a g.l.b.

Corollary 1. For a lattice \mathbb{L} to be conditionally complete, it is sufficient that every bounded non-empty subset of \mathbb{L} has a l.u.b.

Corollary 2. Given a conditionally complete lattice \mathbb{L} , every non-empty subset which has a lower bound has a g.l.b. and dually, every non-empty subset which has a greatest bound has a l.u.b.

Theorem 2. Let \mathbb{L} be a conditionally complete lattice. If elements $0_{\mathbb{L}} = \bigwedge \mathbb{L}$ and $1_{\mathbb{L}} = \bigvee \mathbb{L}$ are added by union to \mathbb{L} then, the result is a complete lattice $\overline{\mathbb{L}} = \mathbb{L} \cup \{0_{\mathbb{L}}, 1_{\mathbb{L}}\}$ which is called the complete lattice extension of \mathbb{L} .

Direct products over lattices maintain certain lattices properties which means that it is possible to build lattice structures from direct products $\mathbb{P} \times \mathbb{Q}$.

Definition 7. *The direct product $\mathbb{P} \times \mathbb{Q}$ of two posets \mathbb{P} and \mathbb{Q} is the set of all pairs (x, y) with $x \in \mathbb{P}$ and $y \in \mathbb{Q}$, partially ordered by \leq , following the rule:*

$$(x_1, y_1) \leq (x_2, y_2) \iff x_1 \leq_{\mathbb{P}} x_2 \text{ and } y_1 \leq_{\mathbb{Q}} y_2 \quad (2.1)$$

Theorem 3. *The direct product of any two lattices is a lattice.*

Corollary 3 (Product Lattice). *Given lattices $\{\mathbb{L}_1, \dots, \mathbb{L}_n\}$, a binary relation over their product $\mathbb{L}_1 \times \dots \times \mathbb{L}_n$ is defined by $(x_1, \dots, x_n) \leq (y_1, \dots, y_n) \iff x_i \leq y_i$, for $i = 1 \dots n$ and $x_i, y_i \in \mathbb{L}_i$ form a **product lattice**.*

If \mathbb{L}_i is complete for all $i = 1, \dots, n$, then for all $x^j \in (\mathbb{L}_1 \times \dots \times \mathbb{L}_n)$ with $j \in J$:

$$\bigvee_{j \in J} x^j = \left(\bigvee_{j \in J} x_1^j, \dots, \bigvee_{j \in J} x_n^j \right) \text{ and } \bigwedge_{j \in J} x^j = \left(\bigwedge_{j \in J} x_1^j, \dots, \bigwedge_{j \in J} x_n^j \right) \quad (2.2)$$

therefore, its product lattice is complete.

A set of lattices $\{\mathbb{L}_1, \dots, \mathbb{L}_n\}$ is named the set of **constituent lattices**. For a lattice \mathbb{L} and a natural number $n > 0$, we denote n direct product of n -copies of \mathbb{L} using the symbol \mathbb{L}^n .

A lattice can be also defined as an algebraic structure. A lattice is an algebraical structure that comes with two binary operations, denoted by \vee and \wedge and named join and meet, respectively.

Definition 8 (Algebraic lattice). *An algebraic structure $(\mathbb{L}, \vee, \wedge)$ consisting of a set \mathbb{L} and two binary operations \vee and \wedge on \mathbb{L} is called lattice if the following identities holds for any $x, y, z \in \mathbb{L}$:*

Commutative laws.	Associative laws.	Absorption laws.
$x \vee y = y \vee x$	$x \vee (y \vee z) = (x \vee y) \vee z$	$x \vee (x \wedge y) = x$
$x \wedge y = y \wedge x$	$x \wedge (y \wedge z) = (x \wedge y) \wedge z$	$x \wedge (x \vee y) = x$

The idempotent laws $x \vee x = x$ and $x \wedge x = x$ arise from the absorption laws. It is easy to verify that a lattice consist of two commutative monoids (\mathbb{L}, \vee) and (\mathbb{L}, \wedge) interacting appropriately. Moreover, a bounded lattice satisfies the identity laws, i.e., $x \vee 0_{\mathbb{L}} = x$ and $x \wedge 1_{\mathbb{L}} = x$ for any $x \in \mathbb{L}$.

Lattice structures play an important role in universal algebra because meet and join are commutative and associative, so lattice can be understood as two semigroup structures with the same domain.

On the one hand, if (\mathbb{L}, \leq) is a lattice, then the partial order on a lattice \mathbb{L} is given by each of the next prescriptions:

$$x \leq y \iff x \wedge y = x; \text{ or } x \leq y \iff x \vee y = y. \quad (2.3)$$

where $x \wedge y$ and $x \vee y$ are defined for $x, y \in \mathbb{L}$.

On the other hand, $x \wedge y$ and $x \vee y$ are given by the infimum and supremum operations. Therefore, the algebraic definition 8 and definition 4 are equivalent.

Mappings between lattice have meaningful richness and they can be explored from the notions below. A mapping ϕ of a poset \mathbb{P} into a poset \mathbb{Q} is order preserving if for any $x, y \in \mathbb{P}$ such that $x \leq y$, then $\phi(x) \leq \phi(y)$. An order preserving mapping is also called **isotone**.

Lemma 2. *In any lattice (\mathbb{L}, \leq) , the operations join and meet are isotone:*

$$\text{if } y \leq z, \text{ then } x \wedge y \leq x \wedge z \text{ and } x \vee y \leq x \vee z. \quad (2.4)$$

Lemma 3. *Any lattice (\mathbb{L}, \leq) satisfies the following distributive inequalities*

$$x \vee (y \wedge z) \leq (x \vee y) \wedge (x \vee z) \quad (2.5)$$

$$x \wedge (y \vee z) \geq (x \wedge y) \vee (x \wedge z) \quad (2.6)$$

Lemma 4. *Any lattice (\mathbb{L}, \leq) satisfies the following modular inequality:*

$$x \leq z \implies x \vee (y \wedge z) \leq (x \vee y) \wedge z. \quad (2.7)$$

Definition 9 (Lattice morphisms). *Let \mathbb{L} and \mathbb{M} be lattices, a mapping $\psi : \mathbb{L} \rightarrow \mathbb{M}$ that satisfies $\psi(x \vee_{\mathbb{L}} y) = \psi(x) \vee_{\mathbb{M}} \psi(y)$ and $\psi(x \wedge_{\mathbb{L}} y) = \psi(x) \wedge_{\mathbb{M}} \psi(y)$ is named **lattice homomorphism**. If the lattice homomorphism is bijective (one to one), it is called **lattice isomorphism**.*

*An isomorphism $\psi : \mathbb{L} \rightarrow \mathbb{L}$ from a lattice to itself is called **automorphism**.*

Some remarks are: two lattices \mathbb{L} and \mathbb{M} are isomorphic if and only if there exists an isomorphism between them. A lattice isomorphism ϕ is an **order-embedding** that is, $x \leq y$ if and only if $\phi(x) \leq \phi(y)$ for all $x, y \in \mathbb{L}$. A lattice isomorphism is an automorphism if $\mathbb{L} = \mathbb{M}$. A homomorphism between bounded lattices satisfies $\psi(0_{\mathbb{L}}) = 0_{\mathbb{M}}$ and $\psi(1_{\mathbb{L}}) = 1_{\mathbb{M}}$. Homomorphisms between lattices are isotone and satisfies preservation of limits, this is, preserves infimum and supremum.

The distributive inequality in Lemma 3 is weaker than the distributive law. Since lattices have two binary operations, it is necessary to study whether one of the operations distributes over the other.

Definition 10. *A lattice (\mathbb{L}, \leq) is said to be **distributive** if and only if satisfies the following properties:*

$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z) \quad \forall x, y, z \in \mathbb{L} \quad (2.8)$$

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z) \quad \forall x, y, z \in \mathbb{L} \quad (2.9)$$

Theorem 4. For any lattice, the Properties 2.8 and 2.9 are equivalent.

Note that every chain is a distributive lattice, any sublattice of a distributive lattice is distributive and any product lattice of distributive lattices is distributive.

We can also verify that for any distributive lattice:

$$\text{if } x \wedge z = y \wedge z \text{ and } x \vee z = y \vee z \text{ then } x = y. \quad (2.10)$$

Definition 11. A complete lattice \mathbb{L} is said to be **Brouwerian** if the infinite distributivity property holds, this is:

$$x \wedge \left(\bigvee_{i=1}^n y_i \right) = \bigvee_{i=1}^n (x \wedge y_i) \quad (2.11)$$

Hence every Brouwerian lattice is distributive. This definition is different from the usual definition given in (BIRKHOFF, 1967), but is equivalent to it (ANDERSON; FEIL, 2012).

Complements play a primordial role in the studies of lattices. For instance, any complemented distributive lattice forms a Boolean algebra and for a distributive lattice, the complement of an elements $x \in \mathbb{L}$ when it exists is unique.

Definition 12. By a **complement** of x in a bounded lattice (\mathbb{L}, \leq) is meant a pair of element $x, y \in \mathbb{L}$ such that $x \vee y = 1_{\mathbb{L}}$ and $x \wedge y = 0_{\mathbb{L}}$. The lattice \mathbb{L} is said to be **complemented** if all its elements have complement.

If the complement is unique, we write $\neg x = y$ and equivalently, $\neg y = x$. The unary operation over \mathbb{L} , called complementation, introduces an analogue of logical negation into lattice theory. Nevertheless, in lattice theory, the term negation is reserved for another specific class of operator.

Definition 13. An operator $\nu : \mathbb{L} \rightarrow \mathbb{L}$, where \mathbb{L} is a complete lattice is called **negation** or **conjugate** if it fulfill the following conditions: $\nu(\nu(x)) = x$ and $x \leq y$ implies $\nu(y) \leq \nu(x)$.

Note that Definition 13 assumes that the operator ϕ is order reversing and an involution. From these properties, it is possible to prove that $\nu(0_{\mathbb{L}}) = 1_{\mathbb{L}}$ and $\nu(1_{\mathbb{L}}) = 0_{\mathbb{L}}$. Particularly, a dual automorphism ν is named negation if ν is an involution.

Definition 14. An element x of a lower bounded lattice \mathbb{L} is an **atom** if $0_{\mathbb{L}} < x$ and there is no $y \in \mathbb{L}$ such that $0_{\mathbb{L}} < y < x$. The lattice \mathbb{L} is called **atomic** if for every $z \neq 0_{\mathbb{L}}$, there exists an atom $x \in \mathbb{L}$ such that $x \leq z$ and is called **atomistic lattice** if every element $z \in \mathbb{L}$ is a least upper bound of a set of atoms.

Co-atom, co-atomic lattice and **co-atomistic lattice** notions can be defined by duality.

Intervals are a natural and frequent option to describe closed sets. As we seen before a convex sublattice of a lattice can be described in terms of closed intervals. In this section, we review the so-called lattice of closed intervals.

Theorem 5 (Lattice of intervals). *The class of closed sub-intervals of a lattice \mathbb{L} , $\mathbb{I} = \{[\underline{w}, \bar{w}] : \underline{w}, \bar{w} \in \mathbb{L}\}$ together with the binary inclusion relation:*

$$\mathbf{x} \leq_{\mathbb{I}_L} \mathbf{y} \iff \mathbf{x} \subseteq \mathbf{y}, \text{ for any } \mathbf{x} = [\underline{w}, \bar{w}], \mathbf{y} = [\underline{y}, \bar{y}] \in \mathbb{I}_L \quad (2.12)$$

is a meet-sublattice of $\mathcal{P}(\mathbb{L})$. The meet is given by:

$$[\underline{w}, \bar{w}] \wedge_{\mathbb{I}_L} [\underline{y}, \bar{y}] = [\underline{w}, \bar{w}] \cap [\underline{y}, \bar{y}] \quad (2.13)$$

Even more, a standard join operator can be defined for \mathbb{I}_L :

$$[\underline{w}, \bar{w}] \vee_{\mathbb{I}_L} [\underline{y}, \bar{y}] = [\underline{w} \wedge_{\mathbb{L}} \underline{y}, \bar{w} \vee_{\mathbb{L}} \bar{y}] \quad (2.14)$$

We will refer to the lattice $(\mathbb{I}_L, \wedge_{\mathbb{I}_L}, \vee_{\mathbb{I}_L})$ as **lattice of closed intervals** or, shortly, lattice of intervals. As an observation, a closed interval $[x, x]$ whose lower and upper bounds are equal is an atom. We denote the set of atoms of \mathbb{L} using the symbol $a(\mathbb{I}_L)$.

Theorem 6 (The Completeness of the Lattice of Intervals). *Given a lattice \mathbb{L} , its lattice of intervals $(\mathbb{I}_L, \leq_{\mathbb{I}_L})$ forms a lattice with $0_{\mathbb{I}_L} = \emptyset$ if the lattice \mathbb{L} is bounded then \mathbb{I}_L has an upper-bound given by $1_{\mathbb{I}_L} = [0_{\mathbb{L}}, 1_{\mathbb{L}}]$. Moreover, if \mathbb{L} is a complete lattice then, its lattice of intervals is also complete.*

Closed intervals are the geometrical base of some lattice based applications, for example: hyperboxes driven classifiers (PEDRYCZ; PARK; OH, 2008).

Definition 15. *Given two lattices \mathbb{L} and \mathbb{M} , a mapping $\phi : \mathbb{L} \times \mathbb{I}_L \rightarrow \mathbb{M}$ together with some $r_\phi \in \mathbb{M}$ that satisfies the following conditions:*

1. $\phi(z, [\underline{w}, \bar{w}]) \geq r_\phi$ if and only if $z \in [\underline{w}, \bar{w}]$.
2. If for all $y, z \in \mathbb{L}$, $[\underline{w}, \bar{w}] \in \mathbb{I}_L$ such that $y \leq z \leq \underline{w}$ then, $\phi(y, [\underline{w}, \bar{w}]) \leq \phi(z, [\underline{w}, \bar{w}])$ where $y, z \in \mathbb{L}$.
3. If for all $y, z \in \mathbb{L}$, $[\underline{w}, \bar{w}] \in \mathbb{I}_L$ such that $\bar{w} \leq y \leq z$ then, $\phi(z, [\underline{w}, \bar{w}]) \leq \phi(y, [\underline{w}, \bar{w}])$ where $y, z \in \mathbb{L}$.

is said to be an \mathbb{I}_L -set descriptor. In this case r_ϕ is called the r -cut of ϕ .

As aforementioned, lattice-based classifiers such as fuzzy lattice neural networks, MPs and θ -FAMs use closed intervals to perform classification. Specifically, MPs are

based on algebraic mathematical morphology which in turn have complete lattices as its framework. Therefore, to analyze the behavior of \mathbb{L} -set descriptors is essential to characterize the described sets. Given an \mathbb{L} -set descriptor ϕ , the mapping $H_\phi : \mathbb{L} \rightarrow \mathcal{P}(\mathbb{L})$ for any interval $[\underline{w}, \bar{w}]$ is represented by:

$$H_\phi([\underline{w}, \bar{w}]) = \{z \in \mathbb{L} : \phi(z, [\underline{w}, \bar{w}]) \geq r_\phi\} \quad (2.15)$$

where r_ϕ is the r -cut of ϕ . This link between sets and \mathbb{L} allows us to check if two different \mathbb{L} -set descriptors describe the same values.

Definition 16. *Two \mathbb{L} -set descriptors ϕ_1 and ϕ_2 are isomorphic if and only if, for any $[\underline{w}, \bar{w}]$, holds:*

$$H_{\phi_1}([\underline{w}, \bar{w}]) = H_{\phi_2}([\underline{w}, \bar{w}])$$

2.3 Notions on Lattice Ordered Groups and Rings

Many interesting research of lattice theory concern lattice ordered structures that have an addition, a multiplication or some other appropriate mapping. Some examples are: lattice ordered groups, ordered monoids, lattice vector spaces, ordered rings and fields. Here, we review two lattice ordered structures: ℓ -groups and ℓ -rings. The first one concerns groups which are at the same time lattices with certain properties. Some basic equalities and concepts are presented and abelian groups are characterized. Then, we discuss the completeness of ℓ -groups and ℓ -rings. We conclude this section connecting the algebraic operations presented here and the \mathbb{L} -set descriptors described previously.

We will use the additive notation throughout this work, a group translation will be written $x \rightarrow z + x + z$, the inverse of x will be denoted using the symbol $-x$ and the group identity will be denoted e . For practical purposes, sometimes we will set $e = 0$, following the standard group notation.

A group is called lattice ordered group if its translations are order preserving, formally:

Definition 17. *A group $(\mathbb{G}, +)$ such that (\mathbb{G}, \leq) is also partially ordered and satisfies:*

$$x \leq y, \implies z + x + z \leq z + y + z \quad (2.16)$$

*is said to be a **po-group**.*

A po-group \mathbb{G} , which is at the same time a lattice is called lattice-ordered group or **ℓ -group**. Furthermore, if the relation \leq of an ℓ -group is a total order then, \mathbb{G} is called a totally ordered group or **\mathbf{o} -group**, for short. A commutative ℓ -group is called **abelian ℓ -group**.

An immediate result of the Definition 17 is stated below:

Lemma 5. *In any ℓ -group \mathbb{G} , every group translation is an order automorphism.*

Therefore, for any po-group \mathbb{G} , the additive inverse is an dual automorphism of the underlying poset.

The set $\mathbb{G}^+ = \{x \in \mathbb{G} : 0 \leq x\}$ is named the **positive cone** of the po-group \mathbb{G} . An element x in an po-group \mathbb{G} is a **positive element** if $0 \leq x$ (strictly positive if $0 < x$) and, for any element $x \in \mathbb{G}$, the **positive part** x^+ of x is given by $x \vee 0$ and the **negative part** $x^- = -x \vee 0$. For any po-group \mathbb{G} the following statements are true:

$$x \leq y \implies -z + x + z \leq -z + y + z \quad (2.17)$$

$$x \leq y \implies y - x \in \mathbb{G}^+ \quad (2.18)$$

$$x \leq y, z \leq w \implies x + z \leq y + w \quad (2.19)$$

Theorem 7 (Theorem 2.1.1. of (STEINBERG, 2010)). *If \mathbb{G} is a po-group with a positive cone \mathbb{G}^+ , then*

1. $\mathbb{G}^+ + \mathbb{G}^+ \subseteq \mathbb{G}^+$
2. $-x + \mathbb{G}^+ + x \subseteq \mathbb{G}^+$
3. $\mathbb{G}^+ \cap \mathbb{G}^- = e$
4. $x \leq y \iff y - x \in \mathbb{G}^+$

for all $x, y \in \mathbb{G}^+$.

Conversely, if \mathbb{G} is a group and there exists a normal subsemigroup S that satisfies 7.3 then $x \leq y \iff y - x \in \mathbb{G}^+$ is a partial order of \mathbb{G} which makes \mathbb{G} into a po-group with positive cone $\mathbb{G}^+ = S$.

Theorem 7 demonstrates that the partial orders of any po-group are in one to one correspondence with the positive cone of \mathbb{G} . In fact, the positive cone \mathbb{G}^+ is usually referred as the partial order of \mathbb{G} .

A pair of elements in the po-group \mathbb{G} are comparable exactly when their difference is comparable to zero, thus if $\mathbb{G} = \mathbb{G}^+ \cup \mathbb{G}^-$ then \mathbb{G} is totally ordered. So, elements of any o-group \mathbb{G} satisfy trichotomy: every element $x \in \mathbb{G}$ is either positive, negative, or zero. Furthermore, If an o-group \mathbb{G} is not trivial (i.e. $|\mathbb{G}| \neq 1$), then its positive cone is infinite, since for all $x \in \mathbb{G}^+$, $x \neq 0$ then $x + x \neq x$. In general, for $x, y \in \mathbb{G}$, where \mathbb{G} is a po-group satisfy the equation:

$$-(x \wedge y) = -x \vee y = [0 \vee (-y + x)] - x \quad (2.20)$$

whenever these three expression exists. Particularly, a po-group \mathbb{G} is an ℓ -group if and only if $x \vee 0$ ($x \wedge 0$) exists for every $x \in \mathbb{G}$. Excluding the trivial case, an ℓ -group cannot

be a bounded lattice and therefore can not be a complete lattice in the usual sense. A complete ℓ -group extension $\mathbb{G} \cup \{+\infty, -\infty\}$ can be defined as extension of an ℓ -group \mathbb{G} where \mathbb{G} is conditionally complete and $\{\pm\infty\}$ denotes the bounds of \mathbb{G} .

Recalling, for conditionally complete lattices the existence of supremum and infimum of finite subsets is guaranteed. Some properties arise naturally in this context, for instance:

Theorem 8 (Theorem 2.1.3.(a) of (STEINBERG, 2010)). *Every conditionally complete ℓ -group \mathbb{G} is Brouwerian and hence is distributive.*

Besides, any po-group which is a meet-semilattice (or join-semilattice) is an ℓ -group. From now on, we are especially concerned with ℓ -groups. In this sense, the lattice operations and the group inverse of any ℓ -group satisfies the De Morgan's laws:

Proposition 1 (Proposition 1.1.1 of (ANDERSON; FEIL, 2012)). *Let \mathbb{G} be an ℓ -group then for any $x, y \in \mathbb{G}$, we have.*

$$-(x \wedge y) = -x \vee -y \quad (2.21)$$

$$-(x \vee y) = -x \wedge -y \quad (2.22)$$

A pair $x, y \in \mathbb{G}$ is **disjoint** if and only if $x \wedge y = 0$. Any element of an ℓ -group can be represented by the difference of two disjoint positive elements. This fact and the uniqueness of the representation come from the following equations:

$$x + x^- = x^+ \quad (2.23)$$

$$x = x^+ - x^- \quad (2.24)$$

$$x^+ \wedge x^- = 0 \quad (2.25)$$

And moreover, if every element x of a po-group \mathbb{G} can be uniquely represented as the difference of disjoint positive elements then \mathbb{G} is an ℓ -group. In addition, Equation 2.25, implies that disjoint elements commute.

The absolute value of an element in some structure may be viewed as the distance between the element and zero. In terms of ℓ -groups, the absolute value of any element can be defined in terms of its positive and negative part.

$$|x| = x^+ + x^- \quad (2.26)$$

Note that $|x| = x^+ \vee x^- = x \vee -x$ and like usual absolute value $|x| = x$ whenever $x \in \mathbb{G}^+$. In addition, the absolute value satisfies a weak triangle inequality:

$$|x + y| \leq |x| + |y| + |x| \quad (2.27)$$

Besides, if $x, y \in \mathbb{G}$ are disjoint then $|x| \wedge |y| = 0$. An abelian ℓ -group \mathbb{G} satisfies the triangle inequality.

Theorem 9. *Let \mathbb{G} be an abelian ℓ -group, then for any $x, y \in \mathbb{G}$ we have:*

$$|x + y| \leq |x| + |y| \quad (2.28)$$

Conversely, if \mathbb{G} is an ℓ -group and for any $x, y \in \mathbb{G}$ Equation 2.28 holds true, then \mathbb{G} is an abelian ℓ -group.

The first part of Theorem 9 was stated in p.309 of (BIRKHOFF, 1987) and the converse statement was proved in (KALMAN, 1960). The notation nx (where $n \in \mathbb{N}$ is a natural number) stands for the group sum of n copies of x . Recalling that $x \in \mathbb{G}$ is **torsion element** if exists a positive integer n such that $nx = \sum_{i=1}^n x = 0$, where 0 denotes the identity element of the group and m a positive integer. If \mathbb{G} has not torsion elements besides the identity element, \mathbb{G} is called a **torsion free group**.

Proposition 2 (Proposition 3.5 of (DARNEL, 1994)). *Let \mathbb{G} be an ℓ -group then the underlying group of \mathbb{G} is torsion free.*

In (GLASS, 1999) it is shown that the converse is not true and therefore, the class of torsion-free groups does not coincide with the class of groups that can be made into ℓ -groups. Nevertheless, the torsion free property does imply ℓ -group orderability in the abelian case.

Theorem 10 (Proposition 1.1.7 of (ANDERSON; FEIL, 2012)). *For an abelian group \mathbb{G} , the following statements are equivalent:*

1. \mathbb{G} is torsion free.
2. \mathbb{G} admits an o -group structure.

Consequently, any torsion free abelian group can be converted into an ℓ -group. Note that totally ordered ℓ -groups are not necessarily abelian. Nevertheless, our focus remains in the abelian case.

Abelian groups appear in many areas of study and are one of the most important concepts in abstract algebra, from which many other concepts such as modules and rings are developed, in fact, every ring is by definition abelian with respect to its addition operation. Later, we will check some concepts related with commutativity but first we will review the Archimedean property, a central issue concerning ℓ -groups.

Definition 18. *An ℓ -group \mathbb{G} is an **Archimedean** ℓ -group if for any elements $x, y \in \mathbb{G}$, $nx \leq y$ for any $n \in \mathbb{N}$ implies that $x \leq 0$.*

Note that Definition 18 implies that in Archimedean groups, for any two strictly positive elements $x, y \in \mathbb{G}$ there exist $n, m \in \mathbb{N}$ such that $nx \not\leq y$ and $my \not\leq x$. Therefore Archimedean ℓ -groups are abelian and torsion free.

Theorem 11 (Theorem 2.2 of (ANDERSON; FEIL, 2012)). *Any Archimedean ℓ -group \mathbb{G} is abelian.*

Representation theorems for ℓ -groups have been widely studied. In particular, Archimedean ℓ -groups representations were studied by Bernau's (BERNAU, 1966). The following theorem is a central issue in ℓ -group theory and links totally ordered Archimedean groups to the Archimedean group of reals.

Theorem 12. *The following statements are equivalent for any ℓ -group \mathbb{G} .*

1. \mathbb{G} is an Archimedean o-group.
2. \mathbb{G} is order isomorphic to a subgroup of the real numbers \mathbb{R}

Last theorem is part of Hölder's Theorem. A comprehensive proof of this theorem is given in (STEINBERG, 2010) p.60. In addition, a more modern proof of Hölder's theorem can be found in: (VISWANATHAN, 1969). An immediate result of Theorem 12 is that the unique non-trivial convex ℓ -subgroups of \mathbb{G} are 0 and \mathbb{G} itself. The following example shows that non-Archimedean totally ordered groups are also possible.

Example 1. *Let $\mathbb{G} = \mathbb{R}^2$, given by Cartesian coordinates (x, y) where $x, y \in \mathbb{R}$ ordered by lexicographic order and let the addition operation be the pointwise vector addition. The ordered group \mathbb{G} is abelian and totally ordered but it is not Archimedean.*

Although non-Archimedean o-groups can be embedded in \mathbb{R}^n for some n , they cannot be embedded in the real numbers. Example 1 is the 2-dimensional case. A more dedicated study on these result (Hahn's theorem and Holder's Theorem) can be found in (FLEISCHER, 1981). Non-Archimedean ℓ -groups are beyond the scope of this research. Since Archimedean ℓ -groups are abelian, it is in our best interest to study ℓ -ordered rings in which the underlying additive group is Archimedean. In order to accomplish this goal, we review some basics on the subject:

Definition 19. *A **partially ordered ring**, or **po-ring** $(\mathbb{A}, +, \cdot, \leq)$ for short, is a ring $(\mathbb{A}, +, \cdot)$ with a partial order, in which the following conditions are satisfied:*

$$x \leq y \implies x + z \leq y + z, \quad \forall x, y, z \in \mathbb{A} \quad (2.29)$$

$$0 \leq x \text{ and } 0 \leq y \implies 0 \leq x \cdot y, \quad \forall x, y \in \mathbb{A}^+. \quad (2.30)$$

A po-ring that satisfies $x \wedge y, x \vee y \in \mathbb{A}$, for all $x, y \in \mathbb{A}$ is called **lattice ring** or ℓ -ring for short. If the relation order \leq is a total order, then it is called **totally ordered ring** or o-ring. An ℓ -ring is said to be **commutative** if $x \cdot y = y \cdot x \quad \forall x, y \in \mathbb{G}$. As in the

ℓ -groups case, the partial order of a po-ring can be identified with its positive cone and an ℓ -ring is called Archimedean if its additive ℓ -group is Archimedean (STEINBERG, 2010). The class of interesting po-rings are those with at least one non-trivial multiplication. In particular, ℓ -rings is a rich class of po-rings and not every ring can be made into an ℓ -ring, for example the rational complex numbers. For instance, in ℓ -ring \mathbb{A} we have:

$$\begin{aligned} |x \cdot y| &= |x^+y^+ + x^-y^- - x^+y^- - x^-y^+| \\ &\leq x^+y^+ + x^-y^- - x^+y^- - x^-y^+ \\ &= |x| \cdot |y| \end{aligned}$$

for any $x, y \in \mathbb{A}$ and therefore, the following equation holds:

$$|x \cdot y| \leq |x| \cdot |y| \quad (2.31)$$

A po-ring where the additive group is an Archimedean group with respect to the given order is called Archimedean po-ring. The case where the po-ring is totally ordered and commutative had been widely studied. A question that matter to us is about which especial properties satisfy non-Archimedean and Archimedean commutative o-rings. The answer to this questions is summarized in the following theorems and examples.

Theorem 13. *Let \mathbb{A} be any commutative o-ring, then:*

1. *The underlying additive group of \mathbb{A} is an o-group.*
2. *x is either in the positive cone or in the negative cone of \mathbb{A} .*
3. $0 \leq 1$.
4. *If \mathbb{A} is not trivial, then it is infinite.*
5. $x^2 \in \mathbb{A}^+$

for any $x \in \mathbb{A}$

All the statements in theorem 13 were proved using IsarMathLib, a library for the Isabelle theorem prover, (KOŁODYNSKI, 2018).

Example 2. *Let $\mathbb{A}[\lambda]$ denote the ring of polynomials in one indeterminate λ over the field \mathbb{Q} of rational numbers ordered lexicographically, with the constant term dominating. This is: $x_0 + x_1\lambda + \dots + x_n\lambda^n \geq 0$ iff $x_0 > 0$, or $x_0 = 0$ and $x_1 > 0$ or, ..., or $x_0 = x_1 = \dots = x_{n-1} = 0$ and $x_n \geq 0$. Then $\mathbb{A}[\lambda]$ is a commutative o-ring but it is not Archimedean.*

Last example leads us to an interesting observation: not any commutative o-ring is Archimedean. In addition, the following example shows that there exist Archimedean ℓ -rings which in fact are not totally ordered.

Example 3. Let X be a or Hausdorff space (or T_2 space) (ARKHANGEL'SKII, 1990), and $\mathcal{C}(X)$ the space of all continuous and real valued functions on X . Then $\mathcal{C}(X)$ is an Archimedean ℓ -ring not totally ordered, under the point-wise operations:

- $[f + g](x) = f(x) + g(x)$
- $[f \cdot g](x) = f(x) \cdot g(x)$
- $[f \wedge g](x) = f(x) \wedge g(x)$ and $[f \vee g](x) = f(x) \vee g(x)$

for any $x \in X$ and $f, g \in \mathcal{C}(X)$

However, just as in the previously studied case of ordered groups, we are especially interested in totally ordered rings which are also Archimedean. The following statements summarize some properties of Archimedean o-rings (STEINBERG, 2010).

Theorem 14. Let \mathbb{A} be an Archimedean o-ring then:

1. \mathbb{A} can be embedded into the \mathbb{R} ring.
2. \mathbb{A} is commutative.

Note that Theorem 14 implies that the properties presented in Theorem 13 remain valid and it is also a consequence of Hölder Theorem (STEINBERG, 2010). Examples of Archimedean o-groups and o-rings include: The sets of integers, rationals, reals and even numbers together with the usual addition, multiplication and ordering. Nevertheless, most of the practical applications, datasets and measures found in literature are described in terms of \mathbb{R} .

2.4 Bounded Extensions of ℓ -groups and ℓ -rings

In this section we will review some notions of the complete extension of an ℓ -group with a underlying conditionally complete lattice. Besides, we study a bounded extension on ℓ -rings, which extends its operators.

Note that excluding the trivial case, an ℓ -group cannot be a bounded lattice and therefore can not be a complete lattice in the usual sense. A conditionally complete ℓ -group can be extended to a complete lattice, adding its upper and lower bounds. Some concepts and results relevant to us are listed next:

Definition 20 (Bounded ℓ -group extension). \mathbb{E} is a **bounded ℓ -group extension** of \mathbb{G} if and only if \mathbb{E} is a bounded lattice and $\mathbb{G} = \mathbb{E} \setminus \{-\infty, \infty\}$ is an ℓ -group. If \mathbb{G} is a conditionally complete ℓ -group, \mathbb{E} is a complete lattice, then \mathbb{E} is called **complete ℓ -group extension** (SUSSNER; ESMI, 2011).

Note that a complete ℓ -group extension is an especial case of bounded ℓ -group extensions. In order to make a group extension consistent, the $+$ operator must also be defined in the extended structure. To simplify, we merely consider non-trivial ℓ -groups. Since every group translation is a lattice automorphism, then for any $x \in \mathbb{G}$, we have: $-\infty \leq x \leq \infty$. The addition $+$ of an ℓ -group \mathbb{G} can be extended to $\mathbb{E} = \mathbb{G} \cup \{+\infty, -\infty\}$, using:

$$x + (+\infty) = +\infty + x = +\infty \quad (2.32)$$

$$x + (-\infty) = -\infty + x = -\infty \quad \forall x \in \mathbb{G} \quad (2.33)$$

Two extended operators denoted by $+$ and $+' : \mathbb{E} \times \mathbb{E} \rightarrow \mathbb{E}$ that satisfy the aforementioned Equations 2.32 and 2.33 are considered. The difference between them lies in the following rules:

$$(-\infty) + (+\infty) = (+\infty) + (-\infty) = -\infty \quad (2.34)$$

$$(-\infty) +' (+\infty) = (+\infty) +' (-\infty) = +\infty \quad (2.35)$$

Therefore, bounded ℓ -group extension can be built following Equations 2.32 to 2.33 (SUSSNER; ESMI, 2011). The existence of additive inverse in ℓ -groups is a sufficient condition to define a lattice conjugate in bounded ℓ -group extensions.

Definition 21 (Conjugate). *Given a bounded ℓ -group extension \mathbb{E} of \mathbb{G} , the symbol x^* denotes the **conjugate** of x .*

$$x^* = \begin{cases} -x & \text{if } x \in \mathbb{G} \\ -\infty & \text{if } x = +\infty \\ +\infty & \text{if } x = -\infty \end{cases} \quad (2.36)$$

Note that the mapping $\nu : \mathbb{E} \rightarrow \mathbb{E}$, given by $\nu(x) = x^*$ is a negation on the lattice \mathbb{E} . Hence, a matrix $A \in \mathbb{E}^{n \times m}$ corresponds to a conjugate matrix A^* where each element $a_{ij}^* = [a_{ji}]^*$ and therefore $A = (A^*)^*$. A bounded ℓ -group extension is denoted $(\mathbb{E}, \vee, \wedge, +, +')$ or \mathbb{E} when there is no ambiguity. Some bounded ℓ -group extensions examples are:

- $(\mathbb{R}_{\pm\infty}, \vee, \wedge, +, +')$ and $(\mathbb{Z}_{\pm\infty}, \vee, \wedge, +, +')$ are complete ℓ -group extensions and they are also chains.
- $(\mathbb{R}_{+\infty}^{\geq 0}, \vee, \wedge, \cdot, \cdot')$ where \cdot denotes the extended multiplication.

It can be verified that the underlying additive group of a partially ordered ring is always a partially ordered group, that for an ℓ -ring, its additive group is an ℓ -group and, furthermore, for any o-ring, the underlying additive group is an o-group.

In the same way that there are no bounded and non-trivial ℓ -groups, an ℓ -ring can not

be bounded. Since MM framework are complete lattices and we can complete the lattices when they are conditionally complete, then we propose to extend conditionally complete ℓ -rings, extending its additive ℓ -group and its multiplication \cdot operation:

Definition 22. \mathbb{E} is a **bounded ℓ -ring extension** if and only if \mathbb{E} is a bounded lattice and $\mathbb{A} = \mathbb{E} \setminus \{-\infty, \infty\}$ is an ℓ -ring. In this case, (\mathbb{E} is an extension of \mathbb{A}).

Definition 23. The product \cdot of an ℓ -ring \mathbb{A} can be extended to elements $x \in \mathbb{E}$, as follows:

$$x \cdot (+\infty) = +\infty \cdot x = +\infty \text{ for } 0 < x \quad (2.37)$$

$$x \cdot (+\infty) = +\infty \cdot x = -\infty \text{ for } 0 > x \quad (2.38)$$

$$x \cdot (-\infty) = -\infty \cdot x = -\infty \text{ for } 0 < x \quad (2.39)$$

$$x \cdot (-\infty) = -\infty \cdot x = +\infty \text{ for } 0 > x \quad (2.40)$$

As consequence, we have:

$$(+\infty) \cdot (-\infty) = (-\infty) \cdot (+\infty) = -\infty \quad (2.41)$$

$$(+\infty) \cdot (+\infty) = +\infty \quad (2.42)$$

$$(-\infty) \cdot (-\infty) = +\infty \quad (2.43)$$

In order to be consistent with ℓ -rings, we consider $0 \cdot (-\infty) = 0 \cdot (+\infty) = 0$. Furthermore, addition and additive inverses in a ℓ -ring can be extended using the aforementioned Equations: 2.34, 2.35 and Definition 21, respectively.

An important observation is that for a bounded extension \mathbb{E} presented here, all observations made in the previous sections remains valid for its underlying ℓ -group \mathbb{G} .

2.5 Mathematical Morphology on Complete Lattices

Nowadays, lattice theory is playing an important role in computer science (DAVIDSON; SUN, 1991; RITTER; SUSSNER, 1996; HEIJMANS, 1994; GRAÑA, 2008; GRANA, 2008; VALLE; SUSSNER, 2008b; GANTER; STUMME; WILLE, 2005; GANTER; WILLE, 2012; RITTER; URCID; VALDIVIEZO-N, 2014; VALLE; SOUZA, 2015; SUSSNER; ESMI; JARDIM, 2019; DAN; HU; QIAO, 2020; SUSSNER; CAMPIOTTI, 2020). A specific application is mathematical morphology (MM), which emerged in the mid-1960s and is concerned with the analysis of spatial structures, shapes and forms of objects (SERRA, 1983). It is based, mostly in set theory and integral geometry but its mathematical core is lattice theory. MM is used to research the interactions between certain objects and a chosen structuring element.

There are two viewpoints on MM that have heavily influenced the development of morphological neural networks and their learning algorithms: From the geometrical or topological perspective, MM represents a theory for processing and analyzing objects, i.e., images or

signals, by means of other objects called structuring elements. From the lattice algebraic perspective, MM is a theory of operators on complete lattices (SERRA, 1988; HARALICK; STERNBERG; ZHUANG, 1987; RONSE, 1990).

The basic operators of MM are erosion and dilation, together with its anti-erosion and anti-dilation counterparts (GOUTSIAS; HEIJMANS, 2000).

Definition 24 (Basic morphological operators). *The operators $\varepsilon, \bar{\varepsilon}, \bar{\delta} : \mathbb{L} \rightarrow \mathbb{M}$ between complete lattices are called algebraic **erosion**, **dilation**, **anti-erosion** and **anti-dilation** if for any $Y \subseteq \mathbb{L}$ they, respectively, satisfy:*

$$\varepsilon(\bigwedge Y) = \bigwedge_{y \in Y} \varepsilon(y), \quad (2.44)$$

$$\delta(\bigvee Y) = \bigvee_{y \in Y} \delta(y) \quad (2.45)$$

$$\bar{\varepsilon}(\bigwedge Y) = \bigvee_{y \in Y} \bar{\varepsilon}(y) \quad (2.46)$$

$$\bar{\delta}(\bigvee Y) = \bigwedge_{y \in Y} \bar{\delta}(y) \quad (2.47)$$

It is common to refer to these concept as algebraic morphological operators to avoid confusion with respect to mathematical morphology with structuring elements. A first relationship between erosion and dilation is adjointness.

Definition 25 (Adjunction). *Consider operators $\varepsilon : \mathbb{M} \rightarrow \mathbb{L}$ and $\delta : \mathbb{L} \rightarrow \mathbb{M}$, between complete lattices. The pair (ε, δ) forms an **adjunction** if and only if for all $x \in \mathbb{L}$, $y \in \mathbb{M}$, we have:*

$$\delta(x) \leq y \iff x \leq \varepsilon(y) \quad (2.48)$$

Let $\nu : \mathbb{L} \rightarrow \mathbb{L}$ be an automorphism that reverses the lattice order. The following result establishes a relationship between composition and algebraic morphological operators.

Theorem 15. *Consider an erosion, dilation, anti-erosion and anti-dilation $\varepsilon, \delta, \bar{\varepsilon}, \bar{\delta} : \mathbb{L} \rightarrow \mathbb{M}$ where \mathbb{L} and \mathbb{M} are complete lattice. Given an operator $\psi : \mathbb{M} \rightarrow \mathbb{K}$ where \mathbb{K} is a complete lattice, then the following statements hold:*

1. *if ψ is an erosion then, $\psi \circ \varepsilon$ is an erosion and $\psi \circ \bar{\delta}$ is an anti-dilation.*
2. *if ψ is an dilation then, $\psi \circ \delta$ is a dilation and $\psi \circ \bar{\varepsilon}$ is an anti-erosion.*

An important result shown in (BANON; BARRERA, 1993) implies that any mapping ψ between complete lattice can be expressed as combinations of the operators given in Definition 24. Particularly, any mapping between two complete lattice $\psi : \mathbb{L} \rightarrow \mathbb{L}$ can be expressed as the supremum of erosion and anti-dilations (dually as infimum of

dilations and anti-erosion). Formally, there exists erosions ε^i and anti-dilations $\bar{\delta}^i$ for some index set I such that

$$\psi = \bigvee_{i \in I} (\varepsilon^i \wedge \bar{\delta}^i) \quad (2.49)$$

Duality in MM is a central issue. Negation forms a duality that has been widely studied, some well known results includes:

Theorem 16. *If \mathbb{L} and \mathbb{M} are complete lattices with negations $\nu_{\mathbb{L}}$ and $\nu_{\mathbb{M}}$, respectively and $\varepsilon, \delta, \bar{\varepsilon}, \bar{\delta}$ are an erosion, a dilation, an anti-erosion and anti-dilation, respectively. Then the following statements are true:*

1. *An operator $\bar{\delta}$ is an anti-dilation if and only if $\bar{\delta} = \varepsilon \circ \nu_{\mathbb{L}}$ and $\bar{\delta} = \nu_{\mathbb{M}} \circ \delta$.*
2. *An operator $\bar{\varepsilon}$ is an anti-erosion if and only if $\bar{\varepsilon} = \delta \circ \nu_{\mathbb{M}}$ and $\bar{\varepsilon} = \nu_{\mathbb{M}} \circ \varepsilon$.*

for all $x \in \mathbb{L}$.

The proof of Theorem 16 can be found in (VALLE; SUSSNER, 2008b). Morphological operators over complete ℓ -group extensions have been widely studied. Next, we list a selection of some concepts and results on the topic.

Definition 26 (Matrix products in complete ℓ -group extensions.). *Given a matrix $A \in \mathbb{G}^{m \times p}$ and a matrix $B \in \mathbb{G}^{p \times n}$, the **max-product** $C = A \boxtimes B$ and the **min-product** $D = A \boxdot B$ are given by the following equations:*

$$c_{ij} = \bigvee_{\xi=1}^k a_i^\xi + b_\xi^j \quad \text{and} \quad d_{ij} = \bigwedge_{\xi=1}^k a_i^\xi + b_\xi^j \quad (2.50)$$

Theorem 17 (Morphology and ℓ -group extensions.). *Consider a complete ℓ -group extension \mathbb{G} and a matrix $A \in \mathbb{G}^{n \times m}$, then the operators $\varepsilon, \delta, \bar{\varepsilon}, \bar{\delta} : \mathbb{G}^n \rightarrow \mathbb{G}^m$ given by:*

$$\varepsilon_A(\mathbf{x}) = A^T \boxtimes \mathbf{x} \quad \text{and} \quad \delta_A(\mathbf{x}) = A^T \boxdot \mathbf{x} \quad (2.51)$$

$$\bar{\varepsilon}_A(\mathbf{x}) = A^T \boxdot \mathbf{x}^* \quad \text{and} \quad \bar{\delta}_A(\mathbf{x}) = A^T \boxtimes \mathbf{x}^* \quad (2.52)$$

are respectively an erosion, a dilation, an anti-erosion and an anti-dilation, where A^T denotes the transpose of the matrix A .

In bounded ℓ -group extensions the following statements can be verified using the morphological elementary operators (SUSSNER; ESMI, 2011).

Theorem 18. *Let \mathbb{E} be a bounded ℓ -group extension of \mathbb{G}^n and $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{E}$. The following statements are true:*

1. $\mathbf{z} \leq \mathbf{x} \iff \varepsilon_{\mathbf{v}}(\mathbf{x}) \geq 0$, where $\mathbf{v} = \mathbf{z}^*$ or equivalently if $\bar{\varepsilon}_{\mathbf{w}}(\mathbf{x}) \leq 0$ where $\mathbf{w} = \mathbf{v}$

2. $\mathbf{z} \leq \mathbf{x} \iff \bar{\delta}_{\mathbf{v}}(\mathbf{x}) \leq 0$, where $\mathbf{v} = \mathbf{z}^*$ or equivalently if $\bar{\delta}_{\mathbf{z}}(\mathbf{x}) \geq 0$ where $\mathbf{w} = \mathbf{z}$

In this section, we described some basics on how morphology is applied to bounded ℓ -group extensions. A more detailed study on this subject and the proof of Theorem 18 can be found in (SUSSNER; ESMI, 2011).

2.6 Aggregations and \mathbb{I} -set Descriptors on Archimedean o -Rings

In this section, we review some notions on intervals in ℓ -groups and ℓ -rings. Particularly, we are interested in functions that describe intervals in Archimedean o -groups and o -rings. Also, we are concerned with how to aggregate the information collected by \mathbb{I} -set descriptors whenever an n -dimensional space composed of n individual Archimedean ℓ -groups or ℓ -rings is considered. In order to achieve this analysis, first we focus our attention to \mathbb{I} -set descriptors and their previously studied properties.

Let us observe the following concerning the underlying compatible order of ℓ -groups in Theorem 7 and Definition 15.

Theorem 19. *Let be an ℓ -group \mathbb{G} and the mapping $\phi_{\wedge} : \mathbb{G} \times \mathbb{I}_{\mathbb{G}} \rightarrow \mathbb{G}$ given by:*

$$\phi_{\wedge}(z, [\underline{w}, \bar{w}]) = (z - \underline{w}) \wedge (\bar{w} - z) \quad (2.53)$$

is an $\mathbb{I}_{\mathbb{G}}$ -set descriptor with α -cut $r = 0$.

Proof. there are three properties that we must prove, namely (1),(2),(3) of Definition 15. The first one is evident. (2) Let $y \leq z \leq \underline{w} \leq \bar{w}$ and $\phi_{\wedge}(z, [\underline{w}, \bar{w}]) = z - \underline{w}$ and $\phi_{\wedge}(y, [\underline{w}, \bar{w}]) = y - \underline{w}$. Since $y - \underline{w} \leq z - \underline{w}$, we have: $\phi_{\wedge}(y, [\underline{w}, \bar{w}]) \leq \phi_{\wedge}(z, [\underline{w}, \bar{w}])$. The proof of Property (3) is similar to the proof of (2). \square

We will refer to ϕ_{\wedge} as the inf-descriptor. Although the last theorem applies to any ℓ -group, not every ℓ -group has interesting geometrical properties to perform classification. The intervals graphical representation of commutative o -group showed in Example 1, do not match the graphical representation of hyperboxes illustrated in previous works. Note that the o -group in Example 1 is totally ordered and abelian. Therefore, it becomes necessary to explore which options fit better the models driven by intervals presented in literature (SUSSNER, 1998; KABURLASOS; PETRIDIS, 2000; PEDRYCZ; PARK; OH, 2008). Since Archimedean o -groups can be embedded in \mathbb{R} , they are a viable option to work as theoretical framework. Based on this argument, from now on we will require the underlying structures to be Archimedean and totally ordered.

Theorem 20. *Given an Archimedean o -ring \mathbb{A} , the mapping $\phi_{\times} : \mathbb{A} \times \mathbb{I}_{\mathbb{A}} \rightarrow \mathbb{A}$ given by:*

$$\phi_{\times}(z, [\underline{w}, \bar{w}]) = (z - \underline{w}) \cdot (\bar{w} - z) \quad (2.54)$$

is an $\mathbb{I}_{\mathbb{A}}$ -set descriptor with α -cut $r = 0$.

Proof. (1) Since, $(z - \underline{w})$ and $(\bar{w} - z)$ are both positive if and only if $z \in [\underline{w}, \bar{w}]$, the first property is evident.

(2) Assume $y \leq z \leq \underline{w} \leq \bar{w}$. We have both $(z - \underline{w})$ and $(y - \underline{w})$ are negative or zero, also $(y - \underline{w}) \leq (z - \underline{w})$. On the other hand $(\bar{w} - z)$ and $(\bar{w} - y)$ are positive, and also $(\bar{w} - z) \leq (\bar{w} - y)$. Therefore: $\phi_{\times}(y, [\underline{w}, \bar{w}]) \leq \phi_{\times}(z, [\underline{w}, \bar{w}])$.

The proof of property (3) is similar to the proof of (2). \square

Figure 2 (d) illustrates a visual representation of the Equation 2.54. When necessary, we will refer to ϕ_{\times} as the dot-descriptor.

Theorem 21. *Let \mathbb{A} be an Archimedean o -ring and \mathbb{G} its underlying additive group, ϕ_{\wedge} and ϕ_{\times} be the \mathbb{I} -set descriptors defined by 2.53 and 2.54, respectively. The ϕ_{\wedge} and ϕ_{\times} represent the same set with respect to the mapping $H : \mathbb{I}_{\mathbb{G}} \rightarrow \mathcal{P}(\mathbb{G})$ given by:*

$$H_{\phi}([\underline{w}, \bar{w}]) = \{z \in \mathbb{G} : \phi(z, [\underline{w}, \bar{w}]) \geq 0\}, \text{ for any } [\underline{w}, \bar{w}] \in \mathbb{I}_{\mathbb{G}} \quad (2.55)$$

Proof. Assume $z \in H_{\phi_{\wedge}}([\underline{w}, \bar{w}])$ for an arbitrary interval, then $\phi_{\wedge}(z, [\underline{w}, \bar{w}]) \geq 0$ which means that $(\underline{w} - z)$ and $(z - \bar{w})$ are both positive. Therefore $\phi_{\times}(z, [\underline{w}, \bar{w}]) \geq 0$. The converse can be demonstrated in the same way. \square

It is important to remark that the \mathbb{I} -set descriptors presented here arises naturally from the ordered structures studied in the previous section. Nevertheless, \mathbb{I} -set descriptors may also be defined in other structures.

A question that concerns us is the dimensionality of datasets. A dataset is briefly a collection of data. For example, a dataset lists values for each of the variables such as luminance, weight or pixel value of an object, for each observation of the dataset.

It was also stated that any Archimedean o -group can be extended to an o -ring. Therefore, we assume datasets can be represented by direct products \mathbb{A}^n where \mathbb{A} is an Archimedean o -ring. In particular, datasets are associated with direct products of the Archimedean o -ring \mathbb{R} . Note that the two \mathbb{I} -set descriptors ϕ_{\wedge} and ϕ_{\times} defined above have Archimedean totally ordered co-domains.

The next step is to analyze the properties of previously studied operators in terms of \mathbb{A}^n . In order to describe how to aggregate the information computed by individual \mathbb{I} -set descriptors, we use an especial kind of operator called aggregation.

Definition 27. *Given a bounded lattice \mathbb{L} , an \mathbb{L} -Fuzzy **aggregation function***

$\bigoplus : \mathcal{P}(\mathbb{L}) \rightarrow \mathbb{L}$ *satisfies:*

1. $\bigoplus_{i=1}^1 x = x$ for all $\{x\} \in \mathcal{P}(\mathbb{L})$

2. $\bigoplus_{i=1}^n x_i \leq \bigoplus_{i=1}^n y_i$ for any $\{x_1, \dots, x_n\}, \{y_1, \dots, y_n\} \in \mathcal{P}(\mathbb{L})$ such that $x_i \leq y_i, i = 1, \dots, n$

$$3. \bigoplus_{i=1}^n 0_{\mathbb{L}} = 0_{\mathbb{L}} \text{ and } \bigoplus_{i=1}^n 1_{\mathbb{L}} = 1_{\mathbb{L}}.$$

There exists a large collection of construction methods as well as different important classes of aggregation operators. In particular, aggregation functions classes in the fuzzy context ($L = [0, 1]$) have been widely studied (BORKOTOKEY et al., 2017; DAN; HU; QIAO, 2020) and many of algebraic properties and classes of aggregations have been explored in (KOMORNÍKOVÁ; MESIAR, 2011; CALVO; MAYOR; MESIAR, 2012). The strategy followed here is to study the operations that naturally arise from the topological ordered structures studied in the last sections. In the same direction, we will study operators (supremum, infimum, addition and multiplication) in terms of aggregation functions and \mathbb{L} -set descriptors.

Our first remark is that an Archimedean o-ring multiplication does not satisfy the first property of Definition 27. However, the remaining operations satisfy all the properties required by this definition.

Lemma 6. *Let \mathbb{A} be an Archimedean o-ring, \mathbb{A}^n the direct product of n copies of \mathbb{A} and let \mathbb{E} be the bounded extension of the underlying lattice of the direct product \mathbb{A}^n . The following statements hold:*

1. $\bigwedge_{i=1}^n x_i = x_1 \wedge \dots \wedge x_n$
2. $\bigvee_{i=1}^n x_i = x_1 \vee \dots \vee x_n$
3. $\sum_{i=1}^n x_i = x_1 + \dots + x_n$ (equivalently, $x_1 +' \dots +' x_n$)

for any $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{E}^n$ are aggregation functions.

We will refer to these aggregation functions as inf-aggregation, sup-aggregation and sum-aggregation, respectively. Note that, an aggregation can be used to combine \mathbb{L} -set descriptors applied to individual copies of \mathbb{A} . Then, we define a mapping considering a pair of an aggregation function \bigoplus and \mathbb{L} -set descriptor ϕ as follows:

Definition 28. *Let \mathbb{A} be an Archimedean ℓ -ring and $P = (\bigoplus, \phi)$ be an aggregation-descriptor pair. The **P-descriptor mapping** $\psi : (\mathbb{A}^n, \mathbb{I}_{\mathbb{A}}^n) \rightarrow \mathbb{A}$ is given by:*

$$\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \bigoplus_{i=1}^n \phi(z_i, [\underline{w}_i, \overline{w}_i]) \quad (2.56)$$

The next result, which in fact is a corollary of the last theorem, gives us information about the relation between intervals and P-descriptors.

Corollary 4. Let \mathbb{A} be an Archimedean o-ring and \mathbb{A}^n the direct product of n copies of \mathbb{A} . For any $\mathbb{I}_{\mathbb{A}}$ -set descriptor ϕ and α -cut $r_\phi = 0$, the following property holds for any $z \in \mathbb{A}^n$:

$$\mathbf{z} \in [\underline{\mathbf{w}}, \overline{\mathbf{w}}] \implies \psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \geq 0 \quad (2.57)$$

where ψ is a P -descriptor with $P = (\bigoplus, \phi)$.

Proof. If $\mathbf{z} \in [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$ then, by Definition 15 $x_i = \phi(z_i, [x_i, \overline{x}_i]) \geq 0$ and therefore:

$$\bigoplus_{i=1}^n x_i \geq \bigoplus_{i=1}^n 0 = 0$$

□

In particular, the converse of the statement 2.57 holds for $\bigoplus = \bigwedge$, this is:

$$\mathbf{z} \in [\underline{\mathbf{w}}, \overline{\mathbf{w}}] \iff \bigwedge_{i=1}^n \phi(z_i, [w_i, \overline{w}_i]) \geq 0 \quad (2.58)$$

. On the other hand, the aggregation $\bigoplus = \bigvee$ satisfies:

$$\bigvee_{i=1}^n \phi(z_i, [w_i, \overline{w}_i]) \geq 0 \iff \exists(j \in \{1, \dots, n\}) \text{ such that } z_j \in [w_j, \overline{w}_j] \quad (2.59)$$

Some properties appear when aggregations P pairs are analyzed as sets.

Formally, given a pair $P = (\bigoplus, \phi)$, we can define the mapping $J^\oplus : \mathbb{I}_{\mathbb{L}} \rightarrow \mathcal{P}(\mathbb{L})$ as follows:

$$J^\oplus([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \{\mathbf{z} \in \mathbb{A}^n : \psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \geq r_\phi\} \quad (2.60)$$

that represent the set generated by an interval using a \mathbb{I} -set descriptor ϕ and an aggregation \bigoplus . We use the notation J_ϕ^\oplus to avoid confusion, when necessary. In particular, the symbols J^\wedge , J^\vee , J^+ denote an aggregation function using \bigwedge , \bigvee and \sum , respectively.

Theorem 22. Consider the $\mathbb{I}_{\mathbb{A}}$ -set descriptors ϕ_\wedge and ϕ_\times . The following equations hold:

$$J_{\phi_\wedge}^\wedge([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = J_{\phi_\times}^\wedge([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \quad (2.61)$$

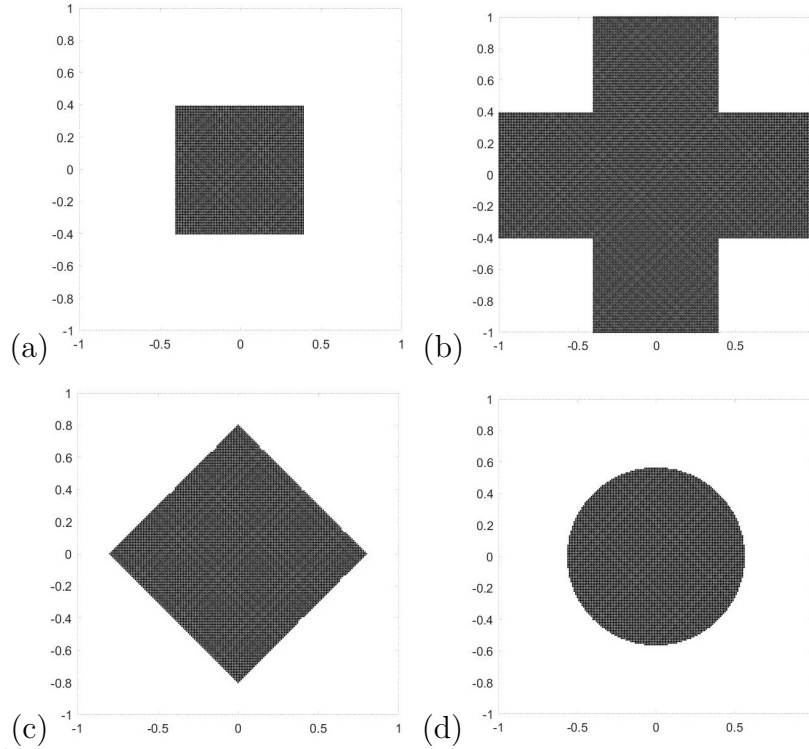
$$J_{\phi_\wedge}^\vee([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = J_{\phi_\times}^\vee([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \quad (2.62)$$

for any $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_{\mathbb{A}}^n$, where \mathbb{A} is an Archimedean o-ring.

Proof. Both statements follow directly from Equations 2.58 and 2.59, respectively. □

Figure 2 illustrates examples of possible aggregation and \mathbb{I} -set descriptors combinations on \mathbb{R}^2 . We will refer to the shapes generated by a pair $P = (\bigoplus, \phi)$ as the geometry of P . The case of sum-aggregations does not fit in Theorem 22. A simple counterexample can be visualized in Figures 2.c and 2.d. Note that the dot-descriptor and

Figure 2 – Generated Sets by Aggregation-Descriptor Pairs



Example in \mathbb{R}^2 for $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] = [(-0.4, -0.4), (0.4, 0.4)]$ using: (a) inf-aggregation (hyperbox) (b) sup-aggregation (c) sum-aggregation and inf-descriptor (d) sum-aggregation and dot-descriptor. The values $\phi(\mathbf{x}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \geq 0$ are colored in black.

inf-descriptor generate different set shapes when the sum-aggregation is considered.

Also, Theorem 22 shows that for the inf-aggregation (dually sup-aggregation) the generated sets by J^\wedge and J^\vee do not depend on \mathbb{I} -set descriptors. Also note that Theorem 22 does not guarantee equal values for different points. In general, it is not true that $\psi_{P_1}(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \psi_{P_2}(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ where $P_1 = (\bigoplus, \phi_\wedge)$ and $P_2 = (\bigoplus, \phi_\times)$.

Theorem 23. *Given an $\mathbb{I}_\mathbb{A}$ -set descriptor ϕ , the following relation is satisfied:*

$$J_\phi^\wedge([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \subseteq J_\phi^+([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \subseteq J_\phi^\vee([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \quad (2.63)$$

for any $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_\mathbb{A}^n$, where \mathbb{A} is an Archimedean o-group.

We already know that for any $\mathbb{I}_\mathbb{A}^n$ -set descriptor ϕ , $\mathbf{z} \in J_\phi^\wedge([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ if and only if $\mathbf{z} \in [\underline{\mathbf{w}}, \overline{\mathbf{w}}]$ and $\mathbf{z} \in J_\phi^\vee([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ if and only if some $i \in 1, \dots, n$ exist such that $\mathbf{z}_i \in [\underline{\mathbf{w}}_i, \overline{\mathbf{w}}_i]$. We know by the duality shown in Lemma 1 that any inf-aggregation can be expressed in terms of a sup-aggregation. Hence, it becomes necessary to study the combinations that involves sum-aggregation and inf-aggregation. Next, we analyze three particular cases of pairs (\bigoplus, ϕ) in terms of distances and other properties given below. But first, we give some definitions that will be used to analyze these mappings. We denote and define the

center of an interval of an Archimedean o-ring $C : \mathbb{I}_{\mathbb{A}}^n \rightarrow \mathbb{R}^n$ as follows:

$$C([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \frac{\underline{\mathbf{w}} + \overline{\mathbf{w}}}{2} \quad (2.64)$$

where the division is by scalar and $+$ denotes the pointwise vector addition.

Definition 29. A real-valued differentiable function f defined on a non-empty convex open set X in the finite-dimensional Euclidean space \mathbb{R}^n is said to be pseudo-convex if, for all $x, y \in X$ such that $\nabla f(x)^T \cdot (y - x) \geq 0$, we have $f(y) \geq f(x)$, where ∇f is the gradient of f , that is:

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)^T \quad (2.65)$$

Convexity and concavity are very well known concepts in several areas of mathematics. We will define these concepts in terms of the Hessian matrix of second partial derivatives. Note that a square matrix $M \in \mathbb{R}^{n \times n}$ is positive semi-definite if for any $\mathbf{x} \in \mathbb{R}^n$ satisfies $\mathbf{x}^T M \mathbf{x} \geq 0$ and negative semi definite if for all $\mathbf{x} \in \mathbb{R}^n$ satisfies $\mathbf{x}^T M \mathbf{x} \leq 0$. Dually is a negative semi-definite matrix if $\mathbf{x}^T M \mathbf{x} \leq 0$ for all $\mathbf{x} \in \mathbb{R}^n$.

Definition 30. A twice continuously differentiable function of several variables is convex on a **convex** set if and only if its Hessian matrix of second partial derivatives is positive semi-definite on the interior of the convex set. Dually, it is **concave** if its Hessian matrix is negative semi-definite under the same conditions.

Inf-Aggregation and Inf-Descriptor Pair

In order to achieve a good analysis of these operations, notions on distance will be used, recalling that Archimedean o-rings can be embedded into the real numbers \mathbb{R} , see Theorem 12.

Let \mathbb{A} be an Archimedean o-ring and assume a $\bigoplus = \wedge$ and ϕ_{\wedge} such that:

$$\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \bigwedge_{i=1}^n \phi_{\wedge}(z_i, [\underline{w}_i, \overline{w}_i]) = \bigwedge_{i=1}^n (z_i - \underline{w}_i) \wedge (\overline{w}_i - z_i) \quad (2.66)$$

Theorem 24. Let \mathbb{A} be an Archimedean o-ring and let ψ be defined by the Equation 2.66. For every $\mathbf{x} \in \mathbb{A}^n$ we have:

$$\mathbf{z} \in J_{\phi_{\wedge}}^{\wedge}([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \implies d_{\infty}(\mathbf{c}, \mathbf{z}) \leq d_{\infty}(\mathbf{c}, \underline{\mathbf{w}}) \text{ (alternatively } d_{\infty}(\mathbf{c}, \mathbf{z}) \leq d_{\infty}(\mathbf{c}, \overline{\mathbf{w}})) \quad (2.67)$$

where $\mathbf{c} = C([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ is the center of the interval $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_{\mathbb{A}}^n$ and d_{∞} denotes the usual ∞ -norm distance.

Proof. Note that $\underline{\mathbf{w}} \leq \mathbf{c} \leq \overline{\mathbf{w}}$ and $d_\infty(\underline{\mathbf{w}}, \mathbf{c}) = d_\infty(\overline{\mathbf{w}}, \mathbf{c}) \geq 0$.

$$\begin{aligned}
\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) - d_\infty(\overline{\mathbf{w}}, \mathbf{c}) &= \bigwedge_{i=1}^n (z_i - \underline{w}_i) \wedge (\overline{w}_i - z_i) - \bigvee_{i=1}^n \overline{x}_i - \frac{\underline{w}_i + \overline{w}_i}{2} \\
&= \bigwedge_{i=1}^n (z_i - \underline{w}_i) \wedge (\overline{w}_i - z_i) + \bigwedge_{i=1}^n \frac{\underline{w}_i + \overline{w}_i}{2} - \overline{w}_i \\
&\leq \bigwedge_{i=1}^n (z_i - \underline{w}_i + \overline{x}_i - \frac{\underline{w}_i + \overline{w}_i}{2}) \wedge (\overline{w}_i - z_i + \overline{x}_i - \frac{\underline{w}_i + \overline{w}_i}{2}) \\
&= \bigwedge_{i=1}^n (z_i - c_i) \wedge (c_i - z_i) \\
&= \bigwedge_{i=1}^n -|z_i - c_i| = -d_\infty(\mathbf{c}, \mathbf{z})
\end{aligned}$$

therefore $\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \geq 0 \implies d_\infty(\mathbf{c}, \mathbf{z}) \leq d_\infty(\mathbf{c}, \overline{\mathbf{w}})$ \square

We already mentioned the differentiability problems of the infimum aggregation. Therefore, neither convex/concave nor pseudo-convex analysis apply to this case.

Sum-Aggregation and Inf-Descriptor Pair

The second pair analyzed is $P = (\sum, \phi_\wedge)$. Let \mathbb{A} be an Archimedean o-ring, in this part we assume the following operator:

$$\psi(\mathbf{x}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \sum_{i=1}^n \phi_\wedge(x_i, [\underline{w}_i, \overline{w}_i]) = \sum_{i=1}^n (x_i - \underline{w}_i) \wedge (\overline{w}_i - x_i) \quad (2.68)$$

First, we need to characterize the set of elements $\mathbf{z} \in \mathbb{A}^n$ that are included in $J_{\phi_\wedge}^\Sigma([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ where $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_{\mathbb{A}}^n$.

Theorem 25. *Let \mathbb{A} be an Archimedean o-ring and let ψ be defined by the Equation 2.68 therefore, for $\mathbf{z} \in \mathbb{A}^n$ we have:*

$$\mathbf{z} \in J_{\phi_\wedge}^\Sigma([\underline{\mathbf{w}}, \overline{\mathbf{w}}]) \iff d_1(\mathbf{c}, \mathbf{z}) \leq d_1(\mathbf{c}, \underline{\mathbf{w}}) \text{ (alternatively } d_1(\mathbf{c}, \overline{\mathbf{w}}) \leq d_1(\mathbf{c}, \overline{\mathbf{w}})) \quad (2.69)$$

where $\mathbf{c} = C([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ is the center of the interval, $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_{\mathbb{A}}^n$ and d_1 denotes the usual 1-norm distance.

Proof. Note that $\underline{\mathbf{w}} \leq \mathbf{c} \leq \overline{\mathbf{w}}$ and $d_\infty(\underline{\mathbf{w}}, \mathbf{c}) = d_\infty(\overline{\mathbf{w}}, \mathbf{c}) \geq 0$.

$$\begin{aligned}
\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) - d_1(\overline{\mathbf{w}}, \mathbf{c}) &= \sum_{i=1}^n (z_i - \underline{w}_i) \wedge (\overline{w}_i - z_i) - \sum_{i=1}^n \overline{w}_i - \frac{\underline{w}_i + \overline{w}_i}{2} \\
&= \sum_{i=1}^n (z_i - \underline{w}_i - \overline{w}_i + \frac{\underline{w}_i + \overline{w}_i}{2}) \wedge (\overline{w}_i - z_i - \overline{w}_i + \frac{\underline{w}_i + \overline{w}_i}{2}) \\
&= \sum_{i=1}^n (z_i - \frac{\underline{w}_i + \overline{w}_i}{2}) \wedge (\frac{\underline{w}_i + \overline{w}_i}{2} - z_i) \\
&= -d_1(\mathbf{z}, \mathbf{c})
\end{aligned}$$

therefore, $\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = d_1(\overline{\mathbf{w}}, \mathbf{c}) - d_1(\mathbf{z}, \mathbf{c}) \geq 0 \iff d_1(\mathbf{c}, \mathbf{z}) \leq d_1(\mathbf{c}, \overline{\mathbf{w}})$. \square

The set of points $\mathbf{z} \in \mathbb{A}^n$ such that $\mathbf{z} \in J_{\phi_\times}^\Sigma$ are characterized in the last theorem. In this case, parameters of individual unit computations are differentiable almost everywhere. We can extend the derivatives of ψ with respect to parameters \underline{w}_i and \overline{w}_i and z_i :

$$\frac{\partial \psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}])}{\partial \underline{w}_i} = \begin{cases} 0, & \text{if } \frac{\underline{w}_i + \overline{w}_i}{2} < z_i \\ -1 & \text{otherwise} \end{cases} \quad (2.70)$$

$$\frac{\partial \psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}])}{\partial \overline{w}_i} = \begin{cases} 1, & \text{if } \frac{\underline{w}_i + \overline{w}_i}{2} < z_i \\ 0 & \text{otherwise} \end{cases} \quad (2.71)$$

$$\frac{\partial \psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}])}{\partial z_i} = \begin{cases} 1, & \text{if } \frac{\underline{w}_i + \overline{w}_i}{2} < z_i \\ -1 & \text{if } \frac{\underline{w}_i + \overline{w}_i}{2} > z_i \\ 0 & \text{otherwise} \end{cases} \quad (2.72)$$

Note that the value 0 is assigned to those places where the derivative is not defined. Therefore, Equations 2.70, 2.71 and 2.72 can be considered extensions of the usual derivatives. Convexity, concavity and pseudo-convexity are other important properties that can be explored. Nevertheless, in this case, it becomes hard since the derivative of ψ is not defined everywhere. Therefore, this analysis does not apply. Note also the division by 2 in the computing of the center of the interval \mathbf{c} and in the derivatives, which means that the aforementioned extended derivatives are defined in \mathbb{R} .

Sum-Aggregation and Dot-Descriptor Pair

Again we assume \mathbb{A} is an Archimedean o-ring and the pair $P = (\sum, \phi_\times)$. The equation analyzed in this part is given by:

$$\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) = \sum_{i=1}^n \phi_\times(z_i, [\underline{w}_i, \overline{w}_i]) = \sum_{i=1}^n (z_i - \underline{w}_i) \cdot (\overline{w}_i - z_i) \quad (2.73)$$

where $\mathbf{z} \in \mathbb{A}^n$, $[\underline{w}_i, \overline{w}_i] \in \mathbb{I}_{\mathbb{A}}^n$.

Next theorem characterize the set generated by $J_{\phi_\times}^\Sigma([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$:

Theorem 26. *Let \mathbb{A} be an Archimedean o-ring and let ψ be defined by the Equation 2.73 therefore for any $\mathbf{z} \in \mathbb{A}^n$ we have:*

$$\psi(\mathbf{z}) \geq 0 \iff d_2^2(\mathbf{c}, \mathbf{z}) \leq d_2^2(\mathbf{c}, \underline{\mathbf{w}}) \text{ (alternatively } d_2^2(\mathbf{c}, \mathbf{z}) \leq d_2^2(\mathbf{c}, \overline{\mathbf{w}})) \quad (2.74)$$

where $\mathbf{c} = C([\underline{\mathbf{w}}, \overline{\mathbf{w}}])$ is the center of an interval $[\underline{\mathbf{w}}, \overline{\mathbf{w}}] \in \mathbb{I}_{\mathbb{A}}^n$ and d_2 denotes the usual euclidean-norm distance.

Proof. Following the same strategy as in the proof of the Theorem 25, we have:

$$\begin{aligned}
\psi(\mathbf{z}, [\underline{\mathbf{w}}, \overline{\mathbf{w}}]) - d_2^2(\overline{\mathbf{w}}, \mathbf{c}) &= \sum_{i=1}^n (z_i - \underline{w}_i) \cdot (\overline{w}_i - z_i) - \sum_{i=1}^n \left(\overline{w}_i - \frac{\underline{w}_i + \overline{w}_i}{2} \right)^2 \\
&= - \sum_{i=1}^n z_i^2 - z_i(\underline{w}_i + \overline{w}_i) + \left(\frac{\underline{w}_i + \overline{w}_i}{2} \right)^2 \\
&= - \sum_{i=1}^n \left(z_i - \left(\frac{\underline{w}_i + \overline{w}_i}{2} \right) \right)^2 \\
&= -d_2^2(\mathbf{z}, \mathbf{c})
\end{aligned}$$

□

Among the pairs studied in this part, the sum-aggregation and dot-descriptor pair has some interesting properties that may be used later. For example, besides differentiable and smooth, we can perform an analysis on pseudo-convexity and convexity (concavity) of its inputs and parameters.

Next the derivatives for $\underline{w}_i^{j,k}$ and $\overline{w}_i^{j,k}$ and x_i :

$$\frac{\partial \tau_k^j(\mathbf{x})}{\partial \underline{w}_i^{j,k}} = x_i - \overline{w}_i^{j,k} \quad (2.75)$$

$$\frac{\partial \tau_k^j(\mathbf{x})}{\partial \overline{w}_i^{j,k}} = x_i - \underline{w}_i^{j,k} \quad (2.76)$$

$$\frac{\partial \tau_k^j(\mathbf{x})}{\partial x_i} = \underline{w}_i^{j,k} + \overline{w}_i^{j,k} - 2x_i \quad (2.77)$$

As the reader may notice the strategy we are following is to analyze each variable independently, going in the same direction, we get the following results:

Theorem 27. *Let ψ be defined as Equation 2.73 where \mathbf{z} and $\overline{\mathbf{w}}$ are constant, then ψ is pseudo-convex.*

It is also pseudo-convex if we consider \mathbf{z} and $\underline{\mathbf{w}}$ as constant.

In fact, the Hessian matrix in both cases is the null-matrix. Moreover, the analysis of ψ with respect to \mathbf{x} as variable lead us to the following result:

Theorem 28. *Let ψ be defined as Equation 2.73 where $\underline{\mathbf{w}}$ and $\overline{\mathbf{w}}$ are constant, then ψ is concave.*

We summarize the properties of the three pairs studied here in Table 1. In this section we defined two \mathbb{I} -set descriptors and three aggregations taking basic operations of Archimedean o-rings. We also studied some properties of these aggregations working together with \mathbb{I} -set descriptors, in terms of valued-functions and in terms of the sets that they generates. Besides, we extended the theory used to build lattice based classifiers

Table 1 – Aggregation-Descriptors Pairs

	Distance	Differentiable	Convex Analysis
$(\bigwedge, \phi_{\wedge})$	Chebyshev	No	No
(\sum, ϕ_{\wedge})	Manhattan	Almost everywhere	No
(\sum, ϕ_{\times})	Euclidean	Yes	Theorems: 27 and 28

Individual properties studied in this chapter.

(specifically morphological based) and we presented with novel results that relates certain operators with distances.

The results studied here will be central issues in Chapter [5](#), in which we propose a framework to describe lattice-based classifiers.

3 Notions on Classification using Backpropagation and Deep Learning.

Just because you're going
forwards. Doesn't mean I'm
going backwards.

Billy Bragg, Life's a Riot with
Spy Vs Spy Album

The first steps toward functional ANNs took place at the beginning of the 1940s, when Walter Pitts and Warren McCulloch wrote a paper on how the neurons might work. The model was presented as a simple electrical circuit network represented by a graph. Later, Donald Hebb pointed out that neural pathways are modified each time they are used. At the end of the 1950s some of the first efforts to simulate a neural network were conducted. In 1958, inspired by the operation of the eye of a fly, Frank Rosenblatt began to work the perceptron. The Rosenblatt perceptron was designed to achieve binary classification on continuous-valued sets. The proposed perceptron computes a weighted sum of the inputs, performs a threshold operation and, as a result, passes a binary response. Developed at the end of the 1960s, adaptive linear elements (ADALINE) and multiple adaptive linear elements (MADALINE) models stood out in the first years of ANNs studies (WIDROW; LEHR, 1990). The Rosenblatt Perceptron is one of the oldest and simplest learning algorithms out there. Both ADALINE and Perceptron are (single-layer) neural network models. However, Some authors consider ADALINE/MADALINE an improvement over the perceptron (WINTER; WIDROW, 1988). It is worth noting that ADALINE and MADALINE models are applied to eliminate echoes on phone lines until today (WIDROW; LEHR, 1990).

Meanwhile, the McCulloch-Pitts model lacked a mechanism for learning which was crucial to be usable in practice. The Rosenblatt perceptron excelled the learning using a mechanism inspired by the work of Donald Hebb. In his book "The organization of behaviour", which inspired Hebbian Theory, it was stated: "When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased" (HEBB, 2005). In this sense, Rosenblatt built a model following a simple logic: increase the weight value if the perceptron output is too low compared with the example target otherwise if the output is too high, decrease the synaptic weights (ROSENBLATT, 1961). Thus, using this rule, an output of any Rosenblatt perceptron

neuron can be described as a simple linear function (A weighted sum) which is squashed by a threshold. In this case, the simple linear function plays the role of synaptic aggregation, meanwhile the threshold plays the role of a neuron activation function. Although, the mathematical and neurologist background behind this model drastically raised the expectation on ANNs, the early success of some ANN led to an exaggeration of its capability and therefore expectations went unfulfilled. These unrealistic affirmations about Rosenblatt perceptron resulted in halting much of the funding destined to ANN research. Until the early 1980s ANNs remained stoned. Moreover, Marvin Minsky and Seymour Papert showed the limitations of the Rosenblatt model in their book *Perceptrons* (MINSKY; PAPERT, 1969).

In 1982, two events converged. On the one hand, John Hopfield presented an ANN related paper to the National Academy of Sciences (HOPFIELD, 1982). The Hopfield's neural network was a novel recurrent ANN designed to work as associative memory. On the other hand, US and Japan joined forces on the Conference on Cooperative/Competitive Neural Networks. As a result, financing began to flow again. Those event led to a second boom of ANNs research, at the beginning of the 1980s, hybrid models were extensively studied. Later, in 1986, Bernard Widrow and Marcian Hoff described a least mean square method (LMS) to train perceptrons (WIDROW; HOFF, 1960). Training by LMS goal is to minimize the error given by the mean squared error over all training patterns. The base of this rule was to follow gradients in order to update, by a rate (learning rate), the set of synaptic weights, so as to minimize the net output and the target value difference. Consequently, the idea of a multilayer perceptron became stronger and more feasible. Independently, David Rumelhart et al., of the Stanford's psychology department came up with an LMS based multilayer perceptron which distributes pattern recognition errors throughout the whole network. The resulting learning algorithm was named backpropagation (BP) and the resultant ANN architecture was a multilayer perceptron (MLP) trained by backpropagation. The general process of Backpropagation was mentioned in (RUMELHART; HINTON; WILLIAMS, 1986), before it was developed and popularized by the same authors a year later (RUMELHART; HINTON; WILLIAMS, 1985). Nevertheless, the technique described by Rumelhart et al., has many predecessors and it was developed and rediscovered over and over in different periods of times (KELLEY, 1960; BRYSON, 1961). Particularly in 1962, Dreyfus published a simple derivation of the chain rule, which may be considered a predecessor of backpropagation modern implementation.

Continuous activation functions became an important part of these models, for example: hyperbolic tangent and sigmoid that were initially used to prove the universal approximation theorem for multilayer perceptrons (CYBENKO, 1989). MLPs trained by BP shown an excellent response in problems of regression and classification, being successfully tested in many applications. MLP neurons are typically aggregated into layers and the first models were designed considering input, hidden and output layers. The input layer

corresponds to the input vector in some cases applies some kind of normalization but usually performs the identity function. The hidden layers have a parametrizable number of neurons and the output layer have a number of neurons equal to the size of the target vector. Broadly speaking, multilayer perceptron (MLP) is a cascade of single-layer perceptrons (RUMELHART; HINTON; WILLIAMS, 1985). This idea considers a layer of input nodes, a layer of output nodes, and one or more intermediate layers. The intermediate layers are called “hidden layers” because they are not directly observable from the systems inputs and the system outputs. MLP with BP algorithm (MPL-BP) trained through LMS, minimizes the least mean square of the error signal. Therefore MLP layers can tackle vastly more complicated problems than just a single layer, also gradient descent can be easily implemented based on the chain rule and matrix algebra (rings). Application on regression and classification of MLP models with one hidden layer and one output layer were successfully applied. Nevertheless, the use of many hidden layers and more complex networks became important cases of study. It is widely accepted to refer to those networks with one hidden layer as shallow networks and to those networks with two or more hidden layers as deep architectures. In fact, the study of deep architectures or deep learning has become a widely researched branch of machine learning.

Deep Learning is a compilation of machine learning algorithms that attempts to model high-level abstract features in data using computational procedures that focus on multiple, iterative non-linear transformations. Deep learning is a class of ANNs that uses multiple layers to extract refined features progressively from raw inputs. For example, a deep neural network based on MLP-BP has at least four layers: an input, an output and, as minimum two hidden layers.

The first networks with many non-linear transformations were published by Alexey Ivakhnenko using the group method of data handling (IVAKHNENKO, b; IVAKHNENKO, a). In 1980, Kunihiko Fukushima introduced the Neocognitron, which is a deep learning model for visual pattern recognition with multiple layers and multiple non-linear transformations (FUKUSHIMA; MIYAKE, 1982). Nevertheless, those models were not related with gradient descent and the training was manually performed using statistics. In 1989, an application to image classification was developed by Yann LeCun et al. In this model, the standard backpropagation algorithm was applied to learn convolution kernels in a deep neural network with the purpose of recognize handwritten numbers on ZIP codes (LECUN et al., 1989). Later, a model with seven layers named the LeNet-5 was introduced and it was applied to image classification and speech recognition (LECUN; BENGIO et al., 1995; LECUN et al., 1998). Then, the use of deep learning was spreading throughout the world and industrial applications to large-scale were successfully implemented (SCHMIDHUBER, 2015).

Nowadays, DL has fueled great strides in a variety of computer vision problems, such as object detection, action recognition, pose estimation, motion tracking, semantic segmentation

and (GOODFELLOW; BENGIO; COURVILLE, 2016; SHORTEN; KHOSHGOFTAAR, 2019). Then, DL has become a promising research subject in several areas of machine learning. Since, deep learning processing is usually related with feature extraction, those models are used for supervised, semi-supervised, unsupervised learning and reinforcement learning. In certain ways, deep learning architectures such as autoencoders, long short-term memory and convolutional neural network evolved from the MLP-BP model. Moreover, the gradient descent optimization framework is widely accepted as the main learning process of most of these architectures (GOODFELLOW; BENGIO; COURVILLE, 2016). Other common examples of modern deep learning architectures are deep belief, convolutional and recurrent neural network (MOHAMED; DAHL; HINTON, 2011; LECUN; BENGIO et al., 1995; WENG et al., 2014).

We will start this chapter with a review of the concepts involved in supervised learning and gradient descent. Besides, we will present some basic notions on classification used throughout the development of this work.

3.1 Supervised Learning in Classification Problems

Supervised learning is the task of learning a function that maps an input to an output from input-output pairs of examples (training dataset). Supervised learning came from the idea that an algorithm can learn from a training dataset, which can be thought of as the teacher (RUSSELL; NORVIG, 2016). Regression and classification are the two main categories involved in supervised learning. Both share the concept of learning by fitting a training dataset.

The main difference between regression and classification is how the output variable is presented. So, meanwhile in classification the output is categorical, in regression the output is numerical and continuous. In this section we focus on classification instead of regression. Classification is the task of learning an operator $h : X \times \theta \rightarrow L$ that maps elements (**observations**) $x \in X$, to one set of finite predefined labels L where $y \in L$ is a **class**. It is called **multi-class classification** when $|L| > 2$. Formally:

Definition 31. *The **target function** h is known as a classification model and the space H of possible operators such that $h \in H$ is called the hypothesis space.*

In these terms, y is a category that the mapping function h predicts. The **parameters** θ , whenever they exist, could be matrices, vectors, numeric or categorical valued (model dependent).

Some examples of classification problems are: Classifying credit card transactions as legitimate or fraudulent. Predicting tumor cells as benign or malignant. Biometrics, land classification in hyperspectral images, handwritten digits classification and so on.

There exists certain rules to create, validate and test a model in supervised fashion.

Definition 32. A set of k example pairs $\mathcal{D} = \{(x^1, y^1), \dots, (x^k, y^k)\}$ such that $x^i \in X$ is an observation (pattern) and $y^i \in L$ is its class label, is called **supervised learning dataset** (dataset for shortly).

Particularly, three disjoint datasets are used in different stages of learning: **Training** (\mathcal{T}), **validation** (\mathcal{V}) and **test** (\mathcal{E}) stages. In the multiple label case, there is no restriction on how many labels an observation can be assigned to. However, we will focus on the case when a single label is assigned to each observation.

A general approach to multi-class classification has the following characteristics:

1. First, the parameters θ of a model h are computed executing a strategy to fit a dataset $\mathcal{T} \subset \mathcal{D}$.
2. In the validation stage observations $\mathcal{V} \subset \mathcal{D}$ are used to evaluate the quality of generalization.
3. The validated model with best characteristics is evaluated using a test dataset $\mathcal{E} \subset \mathcal{D}$.

This process is required by the learning algorithm to achieve a good performance not only for the training data but also for other unseen observations. Thus, the goal of this process is to build a classifier that not only fits the training dataset but classify new records with an acceptable degree of accuracy.

The performance of a classifier is highly dependent on the characteristics of the data to be classified. In that sense, there is no single classifier that performs best on all datasets.

Empirical tests and simulations are required to compare the performance and to find which classifier fits better in some dataset. Numbers of test records that are correctly (**accuracy/performance**) or incorrectly (**error rate**) predicted by the classification model and the **loss function error** are common measures to evaluate a model.

Some terminology must be reviewed in order to define formally the measures used in this work. A **false positive** (FP) occurs when you receive a positive result for a test but you should have received a negative result. A **false negative** (FN) is an error in which you receive a negative result for a test but you should have received a positive one.

In terms of classification, we can suppose that we have a pair $(x, y) \in \mathcal{D}$ where $y \in L$ is the class assigned to the input x . Let $h \in H$ be an operator that is being tested and $h(x) = z$ where $z \in L$ is the label assigned by the model to an input x . On the one hand, if $y \neq z$ we said that occurs a false negative for the class y and a false positive for the class z . On the other hand, when $h(x) = y$ a correct prediction is considered and we said that occurs a true negative (TN) and a true positive (TP) for the class z and class y , respectively. Last, but not least, the condition positive (P) and the condition negative (N) refers, respectively, to the number of real positive cases and real negative cases in the dataset being tested. Note that $P = TP + FN$ and $N = TN + FP$. With this in mind, some measures are

helpful to understand and compare different classification models. Next we list some of them.

1. $TPR = \frac{TP}{P}$. (True positive rate)
2. $TNR = \frac{TN}{N}$. (True negative rate)
3. $FPR = \frac{FP}{N}$. (False positive rate)
4. $FNR = \frac{FN}{P}$. (False negative rate)
5. $PPV = \frac{TP}{TP + FP}$ (Positive predictive value)
6. $NPV = \frac{TN}{TN + FN}$ (Negative predictive value)
7. $ACC = \frac{TP + TN}{P + N}$. (Accuracy)
8. $ERR = \frac{FP + FN}{P + N}$. (Error rate)

It is quite obvious that these statistical measures are dual or complementary by statistical negation. For example, $TPR = 1 - FNR$, $TNR = 1 - FPR$ and $ACC = 1 - ERR$. Accuracy is a simple measure that can be interpreted as how often a model is correct. This single value is used to compare the quality of classification between different models, in an efficient and easy way. One problem of the accuracy measure is that it does not take into account the class distribution and the prior probabilities of a dataset. Therefore, for imbalanced datasets it does not provide useful information about the behavior of a classifier. Many different classification performance measures that deal better with imbalanced datasets have been used in literature. Some examples are, precision, recall and F1-score. Particularly, the F1-score considers the harmonic mean of the precision and recall to take into account the class distribution for binary classification.

Furthermore, the simplicity of the accuracy measure has been criticized because it does not tell us what the model is doing wrong or right. For that purpose there exists the confusion matrix, which is a table that summarize some aforementioned measures. Confusion matrices are valuable visual tools that present class-label and global classification information. Figure 3 shows an example of the configuration of confusion matrices.

Last, but not least, receiver operating characteristic curve (ROC curve) has been developed as a visual tool to understand those classifiers with probabilistic output. In general, in those classifiers, the resulting class is chosen by a winner takes all competition, or by some contest system based on thresholds. ROC curves can plot TPR versus FPR rates for different values of thresholds.

Figure 3 – Confusion Matrix

		Predicted Class		
		Positive	Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN)	Sensitivity $\frac{TP}{(TP + FN)}$
	Negative	False Positive (FP)	True Negative (TN)	Specificity $\frac{TN}{(TN + FP)}$
		Precision $\frac{TP}{(TP + FP)}$	Negative Predictive Value $\frac{TN}{(TN + FN)}$	Accuracy $\frac{TP + TN}{(TP + TN + FP + FN)}$

Simplified organization of a Confusion matrix.

We can quantify how well each test performs by finding the area under the ROC curve (using integration). This allows us to quantitatively compare different tests or classes (and even perform statistical hypothesis testing to determine if there is a statistically significant difference between two tests). Note that this comparison does not require us to determine a threshold in advance - we are in fact comparing the entire test, not just one specific threshold. Moreover, it is also useful as a graphical tool to analyze the classification for different classes.

This section was intended to introduce the multi-class problem, comparison measures and the methodology that will be used to perform experiments in Chapter 6.

It is also needed to describe how the parameters θ of a model will be computed. Next section presents a review of some gradient descent algorithms.

3.2 Introduction to Gradient Descent

In this section, a basic review of gradient descent applied to classification is done. We are going to start with some basic notions and definitions. Next, we explain how gradient descent is applied to classification, explaining some notions of optimization and minimization. Finally, we will introduce the BP algorithm. Throughout this section we lay the groundwork of the specific algorithms used to train the models presented in Chapter 5 and some models used to perform a comparative analysis in the experiments in Chapter 6.

Basic definitions and notions on Gradient Descent.

In vector analysis, the gradient of a multi-variable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ at a point \mathbf{z} is a vector valued operator denoted $\nabla : \mathbb{R}^n \rightarrow \mathbb{R}^n$, whose components are the partial derivatives of f at the point \mathbf{z} :

$$\nabla f(\mathbf{z}) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(\mathbf{z}) \\ \vdots \\ \frac{\partial f}{\partial x_n}(\mathbf{z}) \end{bmatrix}.$$

There are two methods that make use of the gradient to find stationary points: gradient ascent and gradient descent. Gradient descent (GD) is an optimization algorithm used for computing the local of some function. Its strategy is to move iteratively and proportional to the direction of steepest descent as defined by the negative of the gradient. Conversely, the gradient ascent moves to the positive of the gradient. A formal description is given next:

Definition 33. *Given a multi-variable function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ which is defined and differentiable around certain point $\mathbf{z} \in \mathbb{R}^n$, then the value will decrease fastest if one goes from \mathbf{z}^0 in the direction of negative of the gradient $\nabla f(\mathbf{z})$.*

$$\mathbf{z}^* = \mathbf{z} - \gamma \nabla f(\mathbf{z}) \quad (3.1)$$

If $\gamma \in \mathbb{R}_+$ is small enough, then $f(\mathbf{z}^*) \leq f(\mathbf{z})$. The gradient vector at a point \mathbf{z} is a zero vector if and only if \mathbf{z} is a stationary point, otherwise the negative of the gradient vector can be interpreted as the direction and rate of the fastest decrease of f at some point \mathbf{z} . Consider a starting point \mathbf{z}^0 , GD can be described as the iterative application of equation 3.1 to produce a monotonic sequence: $(\mathbf{z}^0, \mathbf{z}^1, \dots, \mathbf{z}^k)$, where:

$$\mathbf{z}^{i+1} = \mathbf{z}^i - \gamma_i \nabla f(\mathbf{z}^i) \quad (3.2)$$

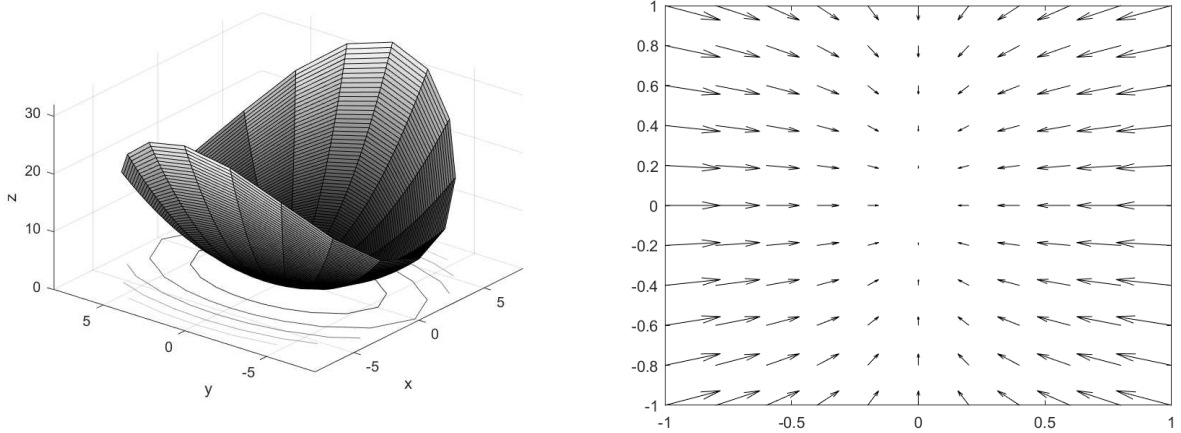
which means that $f(\mathbf{z}_k) \leq f(\mathbf{z}_{k-1}) \leq \dots \leq f(\mathbf{z}^0)$. We will refer to γ_i parameters as learning rates. Learning rates γ play a fundamental role in the study of gradient descent algorithms convergence and some techniques to improve its choice will be studied later.

Gradient descent is a simple and widely researched algorithm that have been extensively used to develop training rules. Next, we detail the general idea behind a classifier trained by gradient descent.

A General Classifier Model Trained by GD

As aforementioned, in supervised learning, the purpose of using SGD is to find the parameters that make a function $h : X \times \theta \rightarrow L$ fit a training dataset. Then, two

Figure 4 – Contour Plots.



(a) Surface and contour plots of the function $f(x, y) = x^2/a + y^2/b$, with constant values $a = 1$ and $b = 3$.
 (b) Velocity plot of the negative gradients $-\nabla f(x, y)$ of the hyper-paraboloid with the same parameters in (a).

types of variables are involved in the computation of h , the dataset observations x and the **learnable parameters** θ . In general, the second ones are directly involved in the gradient descent process, so that learnable parameters change in order to minimize a cost function. For instance, we can take a single element in the training dataset $(x, y) \in \mathcal{T}$ and assume that the output of $h(x, \theta^0) = y^*$, whenever the $y \neq y^*$ the parameters θ^0 must be updated to θ^k using the sequence of updates showed in Equation 3.2 and at certain point we expect to have $h(x, \theta^k) = y$. However, this simple example serves to show the difference between observation variables and learnable parameters.

In order to use gradient descent, it is necessary to use a special kind of function named **loss (or cost) functions** that represents the differences and loss between desired labels and model outputs.

Loss Functions

Many loss functions have been proposed to measure classification model errors. Next, we list some common examples.

Given a training dataset $\mathcal{T} = \{(x^1, y^1), \dots, (x^k, y^k)\}$ such that x^j is an observation and y^j is its assigned label for $j = 1, \dots, k$:

$$\mathcal{L}_{MSE}(\mathcal{T}, \theta) = \frac{1}{k} \cdot \sum_{j=1}^k (h(\mathbf{x}^j, \theta) - y^j)^2 \quad \text{Mean Square Error (MSE)} \quad (3.3)$$

$$\mathcal{L}_{MAE}(\mathcal{T}, \theta) = \frac{1}{k} \cdot \sum_{j=1}^k |h(\mathbf{x}^j, \theta) - y^j| \quad \text{Mean Absolute Error (MAE)} \quad (3.4)$$

$$\mathcal{L}_{CE}(\mathcal{T}, \theta) = \sum_{j=1}^k \sum_{i=1}^{nc} (y_i^j) \cdot \log(h(\mathbf{x}^j, \theta)_i) \quad \text{Cross Entropy (CE)} \quad (3.5)$$

Cross-entropy loss has been widely studied for multi-class classification problems (ROSASCO et al., 2004; MURPHY, 2012). CE measures the performance of models with a probabilistic output. In words, cross-entropy loss value increases as the predicted probability diverges from the real labels y^j . The symbol $\mathcal{L}(S, \theta)$ denotes the loss function for any subset $S \subseteq \mathcal{T}$. Assuming the parameters of the model h are at least differentiable, the most common used strategy to produce this result is to compute the model outputs for each observation x . Then, to compute the gradient vectors for those parameters and, finally, the model is updated. This procedure occurs in the so-called **training stage**. It is important to mention that functions used in classification tasks are assumed to be non-convex. Therefore, the process is done iteratively until the model reaches a local minima that captures the nature of the training dataset. We call a whole iteration over the dataset \mathcal{T} an epoch. However, this produces several limitations and problems related with the GD algorithm that must be reviewed. To this end, next, we study some well known problems and some modification to improve the response of GD algorithms. Hereafter, we assume that the function $h : X \times \theta \rightarrow L$ is differentiable w.r.t. the parameters θ .

Gradient Descent Variants and Improvements

It is important to mention that gradient descent is not a bulletproof algorithm and the differentiability of h is not enough to guarantee the convergence. Though, it is not within the scope of this work, however, we will show some known problems and some techniques to improve the gradient descent algorithm.

The problems of convergence and time needed to train a model are two issues that appear when a gradient descent algorithm is considered. First, although the convexity and pseudo-convexity of functions h where ∇h is Lipschitz continuous guarantee a theoretical convergence to a global solution when a proper λ is chosen (ROBBINS; MONRO, 1951), in practice numerical stability can interfere with the convergence of this algorithm, even when convexity is assumed. Since we assume the function to be optimized is non-convex, the numerical stability can be a major problem and harder to treat. Last, in order to minimize the loss, it is necessary to update the internal learnable parameters for each observation. Therefore, a computational time consuming problem appears when the training dataset and the number of parameters is too big.

The optimal learning rate parameter is a big problem that we have to deal with. If the learning rate is too high, the GD process will skip local minima and sometimes it will never converge. If the learning rate is too low, GD process may never converge because it is trying really hard to exactly find a local minima doing very small steps. The learning rate can affect which minimum you reach and how quickly the local minima is reached (SPALL, 2005).

Here, we will introduce some conventional improvements developed to improve the local minima reached by the GD algorithm, such as the **stochastic gradient descent** (SGD)

and the mini-batch technique. Besides, in order to improve the convergence, we are going to incorporate momentum and other techniques as part of the framework. Last we present learning rate decay which is a basic technique to handle learning rate size problems.

There exist variants of GD, which differ in how much data we use to compute the gradient values of the parameters. The amount of data used in training impacts the quality, accuracy and the time it takes to compute an update of the parameters in gradient descent. Taking this into account, some modifications have been proposed.

Batch Gradient Descent

A first insight is to use the whole training dataset to compute the gradient before compute the update. This modification named **batch gradient descent** (BGD) computes gradients using the entire training dataset.

$$\theta = \theta - \gamma \cdot \nabla \mathcal{L}(\mathcal{T}, \theta) \quad (3.6)$$

where k is the number of observations in the training dataset and $(\mathbf{x}^i, y^i) \in \mathcal{T}$.

The process is very simple: for a predetermined number of epochs, the gradient vector of the loss function h , w.r.t. the parameters is computed considering the whole dataset. Then, we update the learnable parameters in the opposite direction of the resulting gradient. Therefore a unique update is done at each epoch.

Theoretically, if we consider a suitable learning rate, BGD converges to a global minimum when the error surface is convex, or pseudo-convex and, hopefully, to a local minima when the error surface is non-convex.

A very obvious drawback is that a unique update of parameters requires the calculation of gradient for all observations. Hence, BGD can be intractable for datasets that do not fit computational memory.

Single Stochastic Gradient Descent

Both, BGD and **single stochastic gradient descent** (S-SGD) updates to a set of parameters in an iterative manner to minimize an error function are performed. The main difference is that stochastic gradient descent algorithms use a random subset of the training dataset at each iteration of the gradient descent process. Particularly, in S-SGD gradients are computed using a single sample:

$$\theta = \theta - \gamma \cdot \nabla \mathcal{L}((x^i, y^i), \theta) \quad (3.7)$$

where $(x^i, y^i) \in \mathcal{T}$.

S-SGD in contrast to BGD performs a parameter update for each training observation and in each epoch are computed $k = |\mathcal{T}|$ updates. In addition, it is needed to shuffle the training dataset at each epoch.

Mini-Batch Stochastic Gradient Descent

The main difference between **mini-batch stochastic gradient descent** (MB-SGD) and the S-SGD described before is that MB-SGD uses multiple samples to compute gradients. In other words, MB-SGD does a trade-off between learnable parameters updates using the whole dataset and updates using single samples. Hence, MB-SGD takes the best of BGD and S-SGD and performs an update for mini-batches of size n with $n \leq k$:

$$\theta = \theta - \gamma \cdot \nabla \sum_{i=1}^n \mathcal{L}((x^{n(j-1)+i}, y^{n(j-1)+i}), \theta) \quad (3.8)$$

where $j \in \mathbb{N}$ is the number of the batch being processed and $n(j-1) + i \leq k$. The number of samples in each mini-batch is a parameter that is problem dependent. In Chapter 6, we perform a simple experiment to check the behavior. However, we can observe that this algorithm also requires to shuffle the training data set at each epoch.

MB-SGD is typically the algorithm of choice for neural network training. The acronym SGD is usually employed when mini-batches are considered. From now on, we are going to refer to MB-SGD just as SGD. Also, to avoid confusion and for simplicity, we leave out the index $n(j-1) + i$ in the rest of the chapter.

SGD can be impacted by many factor that deserve our attention: The choice of learning rates and oscillations across the gradient directions are two examples treated next.

Learning Rate Schedules

It is clear that a learning rate that is too small leads to a slow convergence or even to a numerical loss of stability and a learning rate that is too big can fluctuate around a local minima, sometimes fluctuate around many local minima or even to diverge. A good practice is to schedule a **learning rate decay** or, in other words, to have a learning rate slowing down as the epoch number increases or as the error starts to decrease. Sometimes both could be considered together.

As an example, let us consider a **decay factor** $\epsilon \in [0, 1]$ that is multiplied by each predefined number of epochs e so that when the SGD gets in advanced stages, the learning rate becomes smaller.

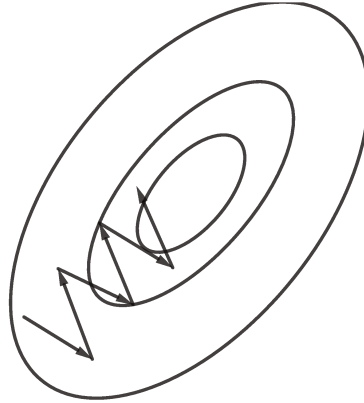
$$\gamma = \gamma - \gamma \cdot \epsilon \quad (3.9)$$

Besides its simplicity, an advantage of this strategy is its low computational cost. However, the introduction of the decay factor parameter and how often the task is scheduled can be a problem. For instance, decay factor values close to 1, or to apply the equation 3.9 many times could reduce the learning factor too much and therefore, to stop the learning.

Momentum

Areas where the surface curves are much different from each other can represent a problem to SGD. Specifically, oscillations can represent a challenge and slow down the

Figure 5 – Oscillations of SGD.



A sequence of steps that follow the opposite direction of the gradient vector. The figure exemplifies oscillations of the optimum search using a contour plot.

search of a local minima. Figure 5 shows a simple example on how oscillations could occur. **Momentum** is a technique developed to accelerate SGD convergence and to improve the search of a local minima, considering a fraction β of previous updates to the current gradient computation. It was first proposed in the seminal paper on backpropagation of Rumelhart et al., (RUMELHART; HINTON; WILLIAMS, 1985). The computation performed considering momentum for a single update is given by:

$$\begin{aligned} v_t &= \beta \cdot v_{t-1} + \gamma \cdot \nabla_{\theta} \mathcal{L}(S, \theta) \\ \theta &= \theta - v_t \end{aligned} \quad (3.10)$$

for any $S \subseteq \mathcal{T}$ and $\beta \in [0, 1]$, so that $\beta = 0$ value means no contribution from the previous step, whereas a value of $\beta = 1$ means maximal contribution from the previous step. This modification of the standard SGD algorithm is motivated from the physical perspective of momentum. Imagine a ball in a hilly terrain rolling down that is going to the deepest of the valley. If the ball is in a part of the hill with a high slope, the ball will gain a lot of momentum energy and will be able to pass through slight hills in its way. The momentum and speed of the ball will decrease as the slope decreases, hopefully the ball will rest at the deepest part of the valley.

Root Mean Square Propagation

Applying the same learning rate to all parameters being updated can be a problem if our dataset is sparse or if our dataset is unbalanced and has more observations from one class than another. Moreover, in general the features of a dataset have different frequencies and standard deviations.

Root Mean Square Propagation (RMSprop) improves the search by using learning

rates that differ for each parameter. Steps follow an average of the element-wise square of the gradient components, for each parameter.

$$v_t = \beta \cdot v_{t-1} + (1 - \beta) \cdot [\nabla_{\theta} \mathcal{L}(S, \theta)]^2 \quad (3.11)$$

where $\beta \in [0, 1]$ is the decay rate of the moving average and $S \subseteq \mathcal{T}$. In order to normalize the update of each individual parameters, the moving average is used as follows:

$$\theta = \theta - \frac{\gamma \cdot \nabla_{\theta} \mathcal{L}(S, \theta)}{\sqrt{v_t} + \epsilon} \quad (3.12)$$

where the division and the square are performed element-wise and the small positive ϵ is used to avoid zero division. RMSprop decreases or increase the learning rates of large or small learnable parameters, respectively. This technique was introduced by Hinton in the context of machine learning ([HINTON; SRIVASTAVA; SWERSKY, 2012](#)).

Adaptive Momentum Estimation

Adaptive Momentum Estimation (ADAM) was first introduced in ([KINGMA; BA, 2014](#)). ADAM considers an adaptive learning rate for each parameter and is considered an improvement of RMSprop. This is, besides it stores an exponential decaying average of past squared gradients, it uses averages of the gradient values and uncentered variances (or second momentum).

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla_{\theta} \mathcal{L}(S, \theta) \\ v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot [\nabla_{\theta} \mathcal{L}(S, \theta)]^2 \end{aligned} \quad (3.13)$$

where S is a mini batch, $\beta_1, \beta_2 \in [0, 1]$ are the decay of the momentum term and the decay of the moving average, respectively. ADAM updates the parameters as follows:

$$\theta = \theta - \frac{\gamma \cdot m_t}{\sqrt{v_t} + \epsilon} \quad (3.14)$$

where division and the square are performed element-wise and ϵ avoid the zero division possibility.

ADAM give priority to those gradient values that are that are similar over many iterations. In addition, if the parameters have much noise, the moving average of the gradient become slower and, therefore, the updates become smaller.

3.3 Deep Learning and Backpropagation

Until now, abstract models and parameters for SGD classification have been considered. Deep Learning is a set of algorithms for machine learning that tries to model abstractions of datasets using computational architectures that support multiple

cumulative non-linear transformations. There is not a unique definition of deep learning but the different definitions have a common base: Multiple layers of non-linear processing units and the progressive extraction of higher level features from a raw input (GOODFELLOW; BENGIO; COURVILLE, 2016).

DL has been applied successfully to supervised learning. In fact, SGD has played a fundamental role in the developing of some DL models. Besides, some deep learning models are based on artificial neural networks and extend the multi-layer perceptron presented in (MAKHZANI; FREY, 2013; SCHMIDHUBER, 2015).

One particular subclass of DL models for multiclass classification are the feedforward DL models based on composition of functions. In this section we provide an abstract definition of a deep feedforward network by means of composition of functions which allow us to divide these kind of models in blocks. In addition, we provide a brief introduction to the BP algorithm. Apart from the description of some functions that are part of this feedforward framework, we present two examples of architectures that use this framework.

Deep Learning Models and Composition of Functions

The deep learning classifier networks considered here are operators $O : X \rightarrow L$ that map a set of observations to a set of labels defined by compositions of m functions:

$$G_\ell(x) = \begin{cases} F_1(x) & : \ell = 1 \\ (F_\ell \circ G_{\ell-1})(x) & : \ell > 1 \end{cases} \quad (3.15)$$

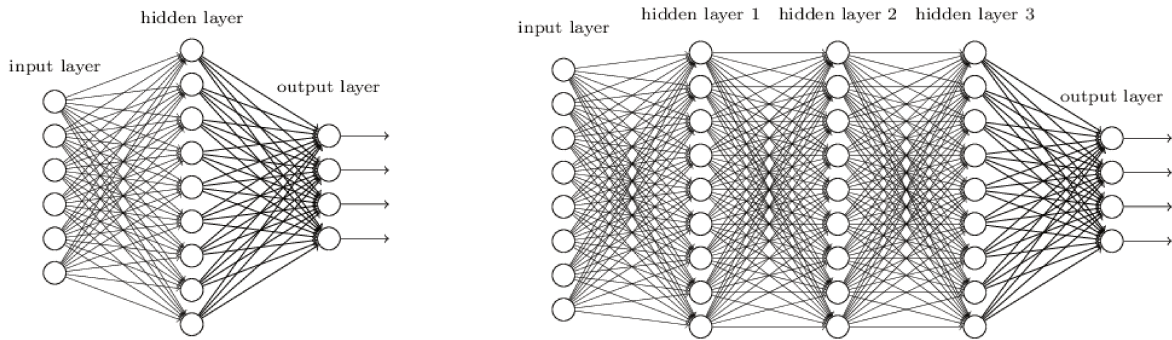
The operator O computes $O(x) = (G_m(x))$ where m is the overall length of the chain, named **the model depth**.

We can divide the parts of a deep learning model defining some operations based on blocks or layers. Particularly, a **layer** corresponds to one function F_ℓ . In other words, the layer ℓ is a block consisting of a set of computational units denoted by f_i^ℓ , that act in parallel where each unit represents a vector-to-scalar function. We can read the notation f_i^ℓ as the i -th computational unit (or node) of the layer ℓ -th. The number of computational units of a layer F_ℓ is called the width of the layer and is denoted $|F_\ell|$.

Each function $f_i^j : \mathbb{R}^d \rightarrow \mathbb{R}$ can be computed, or not, with respect to certain learnable parameters. It is obvious that it is possible to have $F_\ell \neq F_\kappa$ for two layers. Also, it is possible that two computational units of the same layer ℓ compute its outputs w.r.t. different parameters and therefore $f_i^\ell(\mathbf{x}, \theta^{\ell,i}) \neq f_j^\ell(\mathbf{x}, \theta^{\ell,j})$. In order to avoid confusion and repeated indices, we adopt the following notation: $f_j^\ell(\mathbf{x}, \theta) = f(\mathbf{x}, \theta^{\ell,j})$ whenever necessary. Note that the function $F_\ell(\mathbf{x})$ can be regarded as a vector whose components are the unit computations f_i^ℓ for $i = 1, \dots, s$, recalling $s = |F_\ell|$ represents the width of a layer F_ℓ . The notation f_i^ℓ is used as the i -th vector component of F_ℓ .

It was aforementioned the existence of a biological inspiration of the backpropagation algorithms. In this case, each unit of computation resembles a biological neuron in the

Figure 6 – Multilayer Perceptrons.



Abstract MLPs representations.

sense that it receives an input from many other neurons and computes its own activation function which will be sent to many other neurons. The choice of functions F_j to compute these units also is loosely guided by neuroscientist observations about the functions that biological neurons compute.

A deeper biological discussion of multi-layer, feedforward architectures is beyond the scope of this work. A discussion on the biological inspiration of MLP and deep feedforward architectures appeared in (GOODFELLOW; BENGIO; COURVILLE, 2016). Abstract graphical representations of two multilayer models are illustrated in Figure 6.

Each layer can be classified into three different types: **Input layers**, **Hidden layers** and **Loss Output Layer**.

Introduction to Backpropagation

Derivatives and gradient values of compositions involving differentiable functions can be found using multivariable chain rules. In the backpropagation algorithm the gradient is computed using the chain rule, computing the total gradient as a sum of local gradients over all the paths from an the input layer to the output (input-output path). Figure 7 shows a graphic representation of those paths.

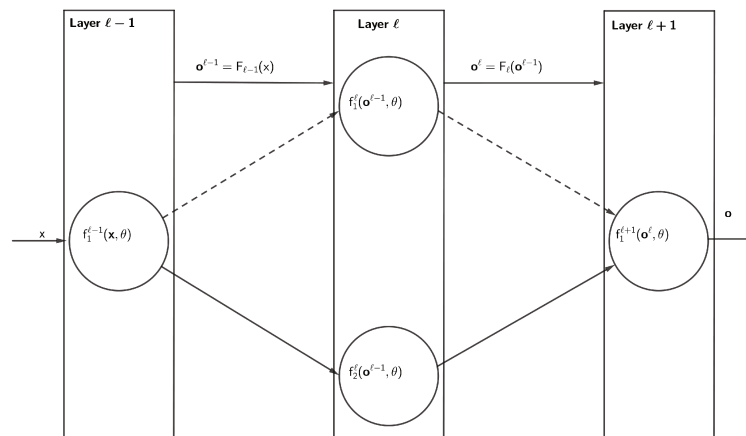
When compositions are considered, a problem is that the loss is given by the composition of parametrizable functions propagated in the entire network and the number of input-output paths can grow exponentially. A better approximation to this computation comes from dynamic programming and parallel computing, (LECUN et al., 1988) and in general is described in terms of two main phases:

1. **Forward phase:** An input set $S \subseteq \mathcal{T}$ is fed into a model O , a forward cascade of computation across the layers using the set of parameters θ of the whole network. By means of a loss function, outputs of O can be compared to the desired output. The

gradient of the obtained loss is computed. Next, the gradient of this loss is computed with respect to the parameters in all layers.

2. **Backward phase:** The goal is to compute the gradient of the loss function with respect to each different parameter, using the chain rule of differential calculus. These gradients are used to update the parameters using SGD. The gradients are computed going from the last layer to the first, in a backward direction. Last, variants of SGD such as those proposed in Section 3.2 can be applied to update parameters.

Figure 7 – Paths between Layers.



A simple network with two paths, three layers and four nodes.

Let us consider a unique node at the layer $\ell - 1$, which is connected with two nodes at layer ℓ and consider a layer $\ell + 1$ also with a unique node which is connected with the two nodes at the layer ℓ . So that there are only two paths from the node $f_1^{\ell-1}$ to the node $f_1^{\ell+1}$. In Figure 7 this example is represented, the dotted and continuous line represents two paths between nodes in layers $\ell - 1$ and $\ell + 1$. The gradients of the parameters $\theta^{\ell-1,1}$ can be computed as follows:

$$\frac{\partial o}{\partial \theta^{\ell-1,1}} = \underbrace{\frac{\partial o}{\partial f_1^\ell} \cdot \frac{\partial f_1^\ell}{\partial f_1^{\ell-1}} \cdot \frac{\partial f_1^{\ell-1}}{\partial \theta^{\ell-1,1}}}_{\text{First path.}} + \underbrace{\frac{\partial o}{\partial f_2^\ell} \cdot \frac{\partial f_2^\ell}{\partial f_1^{\ell-1}} \cdot \frac{\partial f_1^{\ell-1}}{\partial \theta^{\ell-1,1}}}_{\text{Second path.}}$$

In this simple example, we can visualize the process through a network in the backward phase. Nevertheless, aggregate paths to compute a gradient when a larger numbers of nodes is involved, could result in an exponential grow.

The multivariable chain rule is a central definition regarding backpropagation.

Definition 34. Let $w = f(x_1, x_2, \dots, x_m)$ be a differentiable function of m independent variables, and for each $i \in 1, \dots, m$, let $x_i = x_i(t_1, t_2, \dots, t_n)$ be a differentiable function

of n independent variables. Then:

$$\frac{\partial w}{\partial t_j} = \frac{\partial w}{\partial x_1} \cdot \frac{\partial x_1}{\partial t_j} + \dots + \frac{\partial w}{\partial x_m} \cdot \frac{\partial x_m}{\partial t_j} = \sum_1^m \frac{\partial w}{\partial x_m} \cdot \frac{\partial x_m}{\partial t_j} \quad (3.16)$$

for any $j = 1, \dots, n$

The computational graph of a neural network does not have cycles and we can compute gradients using a backward directed strategy by computing first the closest F_k layer to the output layer, then to compute recursively for previous layers in terms of those already computed. The gradient of a deep learning operator O is initialized using the loss function:

$$\nabla(O, \theta) = \frac{\partial \mathcal{L}}{\partial \theta} \quad (3.17)$$

The recursion to compute the gradient of the parameters for a j -th node in a layer ℓ , ∇f_j^ℓ is given, using the multi-variable chain rule, so the strategy to compute the gradient for a certain function or parameter in a layer ℓ using the gradient vector of the layer $\ell + 1$ in a pretty organized way.

Vanishing and Exploding Gradients Problems.

Some of the first challenges faced by BP-based models include slow learning and overfitting. In the first case BP needs thousands of iterations to fit the desired targets and in the second case the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably. Stochastic gradient descent (SGD), mini-batch SGD, gradient descent momentum, studies of dynamics learning rates, learning rate schedules and mini-batch training are some useful techniques to battle slow learning. Meanwhile, validation models try to reduce overfitting.

Other problems linked with gradient descent are vanishing and exploding gradients, first described in (HOCHREITER et al., 2001). These problems arise during training of a MLP-BP with n hidden layers when gradients are being propagated back all the way to the initial layer. Specifically, the vanishing gradient problem occurs when gradients computed from the deeper layers have to go through continuous matrix multiplications because of the chain rule, resulting in gradient values that shrink until they vanish, muddling the update of the parameters and the learning process. In the opposite direction, if the propagated gradients become too large, a numerical crash of the system could occur. Hence, the exploding gradient problem occurs when large gradients values crash the model. There are many techniques to reduce exploding and vanishing problems in DL (BÜHLMANN; GEER, 2011; GLOROT; BENGIO, 2010).

3.4 Some Deep Learning Layers

Backpropagation has been widely discussed in the literature and many well succeeded implementations that consider GPU, parallel computation and dynamic programming exist. Nevertheless, the perspective of this work is to explain a computational model, in which the unitary objects are the layers. In this way, each layer F_ℓ at least performs a forward pass transformation $F_\ell : \mathbb{R}^{|F_{\ell-1}|} \rightarrow \mathbb{R}^{|F_\ell|}$ using inputs coming from the layer $F_{\ell-1}$ and a backward transformation $\nabla F_\ell : \mathbb{R}^{|F_{\ell+1}|} \rightarrow \mathbb{R}^{|F_\ell|}$ using the gradients computed by the backward function of the layer $F_{\ell+1}$. This process allows us to review the operations performed by a layer and to breakdown the process into abstract operations. In this section, we review some concepts and notions on deep learning layers. Specifically, we will see which parameters are involved in some well known layers from the literature. This introductory part is intended for the reader to understand basics on the configuration and operations involved in different layers, so that the reader may be able to use these concepts on some standard implementations of deep learning such as (MATLAB, 2018; CHOLLET et al., 2015; ABADI et al., 2015). Besides, we define the notation used in this text to describe DL networks.

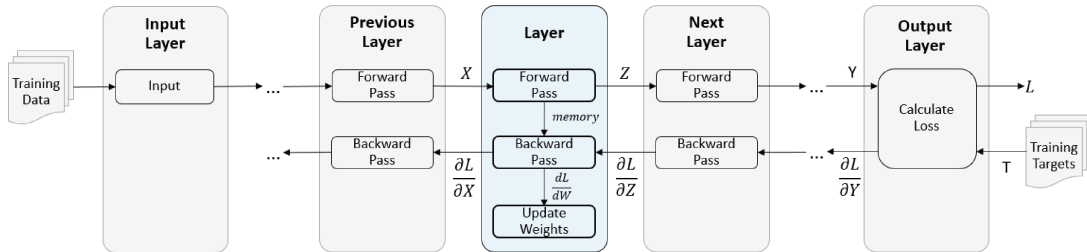
Some Notions on Deep Learning Layer

First, we will define some terms that are involved in the information flow of the backpropagation algorithm.

- **Forward pass function** receives the forward output of a previous layer and computes the output to the next layers.
- **Backward pass function** receives the function of error computed by the next layer and the memory and it is responsible of the update of weights.
- **Forward-backward memory** is the information passed from forward pass to the backward pass function of the same layer.
- **Update weights process** module updates the **learnable parameters** belonging to the layer. Remembering that some layers do not have learnable parameters.
- **Loss function:** Is a statistical, probabilistic or a metric based cost function, i.e., a function to be minimized. Loss functions are discussed in section 3.2.
- **Initialization of parameters** is a function that initialize the required learnable (or not) parameters involved in the computation of a layer.

Figure 8 shows the structure of the computational model. Also, we can distinguish the parts, flows and functions that form a layer.

Figure 8 – Backpropagation Flow Diagram.



Abstract model of a deep learning architecture and its process of training.

Source: (MATLAB, 2018)

- **Input Layers** has a forward pass function. In general, input layers perform normalization, extraction of interest region and some times is desirable to set an input layer to perform identity, i.e. any transformation is performed.
- **Hidden Layers** has, at least, backward pass and forward pass functions. If the layer has learnable parameters it applies updates of those parameters. Hidden layers have been widely studied and there exist many classes of this kind. We give some examples of hidden layers considered in experimental processes of Chapter 6 and in the development of Chapter 5.
- **Output Layers** compute the final output of the model and handle a loss function.

In addition, each layer can have or have not learnable parameters, moreover, each layer can be classified by which task its computations perform, some examples are: Sliding convolutional, fully weighted connected, activation, normalization, pooling and output layer. There exist other types of layers that are beyond of the scope of this work, such as: sequence, cropping, object detection and combination layers (KRIZHEVSKY; HINTON et al., 2009; HE et al., 2016; GLOT; BENGIO, 2010).

Sliding Convolutional Layers

Sliding convolutional layers are used in image and signal processing. The main idea is to perform filter operations using a sliding window over the whole image or signal. These kinds of layers are able to capture spatial and temporal dependency. In other words, sliding layers were developed to understand sophisticated relations between individual

features of data observations. A convolutional layer convolves the input and passes its result to the next layer.

In particular, the convolutional neural network makes use of masks (named also filters or kernels) that are convolved across the data. Then, it applies a cross relation measure between that mask and the slices of data (LECUN; BENGIO et al., 1995).

Convolutional layers were developed to be applied in visual imagery where the data has spatial information and have shown an outstanding accuracy in several image classification problems and are considered hidden layers. The weights of the kernels are involved in the computation of a forward pass function and they work as learnable parameters.

It is not practical to totally connect units of computation to all the inputs, whenever we deal with high-dimensional data with spatial information, such as images. Alternatively, we can connect each unit of computation (neuron) to a local region of input volumes. The size of this connectivity is the size of filter and is called the **receptive field** of the neuron. The depth axis of the connectivity is equal to the depth of the data. Therefore, the connections are local in the space but always totally connected to the entire depth of inputs.

Example 4. *Suppose that the input data has size of $28 \times 28 \times 1$, a grayscale image and assume the filter size is $3 \times 3 \times 1$, so that each unit of computation in the convolutional layer have $3 * 3 * 1 = 9$ parameters plus a bias parameter.*

*Observe that for higher dimensionality such as RGB the filter must have the same size but with a depth 3. In the RGB case the number of parameters will be $3 * 3 * 3 = 27$.*

It is also needed to discuss how many neurons there are in the output volume and how they are organized. The **depth**, **stride** and **zero-padding** are hyper-parameters that help to control the output size of a convolution layer. We discuss these next:

We need to specify the step size (**stride**) with which the filter will slide. For example, when the stride is 2, the filter will jump two pixels at a time. In terms of spatial size, higher values produce smaller outputs volumes.

It is convenient to pad the input data with zeros around the borders. The size of this **padding** filled with zeros is itself a hyper-parameter. In general, it is used to preserve the spatial size of an input. In other words, in order to make the input and output size have same values, we use this zero-padding. Last but not least, we will refer to the set of neurons that are computing the same region of the data as fibres or depth column parameter.

Each output map may combine convolutions with multiple input maps possibly followed by an activation function. Lets consider a set of r filters (column depth size is r) denoted by K_i with a fixed $p \times p$ size and the activation function $\sigma(x) = \frac{1}{1 + e^{-x}}$. For simplicity and symmetry we assume that it is an odd number and $q = \text{ceil}(p/2)$, then a convolutional

layer F_ℓ "simplified" forward pass computes:

$$f_m = \sigma(I * K_m + b^m) \quad (3.18)$$

$$f_m(i, j) = \sigma\left(\sum_{u=-q}^q \sum_{v=-q}^q I(i-u, j-v) \cdot K_m(u, v) + b^m\right) \quad (3.19)$$

where I is an image, $m = 1, \dots, r$, $*$ denotes the convolution operation and i, j are the row and column indices of I pixels. For simplicity, we keep only those parts of the convolution that are computed without pad, so the size of output is spatially smaller than the input, also we are assuming depth of the dataset is one. Nevertheless, these last equations give a general idea of how the forward function could be implemented.

On the other hand, and regarding that the chain rule of a composition simplifies the mathematics, we focus on calculation of the derivatives of the convolution:

$$\frac{\partial f_m(i, j)}{\partial K_m(u', v')} = \frac{\partial}{\partial K_m(u', v')} \sum_{u=-q}^q \sum_{v=-q}^q I(i-u, j-v) \cdot K_m(u, v) + b^m \quad (3.20)$$

since the derivative will be non-zero only when $m = m'$ and $n = n'$, we get:

$$\frac{\partial f_m(i, j)}{\partial K_m(u', v')} = I(i + m', j + n') \quad (3.21)$$

These gradients are used to update the parameters. The backward pass for a convolution operation (for both the data and the weights) is also a convolution (but with the filters flipped). Finally, to compute the backward pass function, we need to consider the connections between layer and the derivative w.r.t. convolution functions. In this case, it is assumed that a convolution layer is followed by a downsampling operation. Next, we do a brief introduction to pooling layers, which apply a downsampling operation. However, there are cases when specific positions and patterns have to be detected and the pooling layer is omitted.

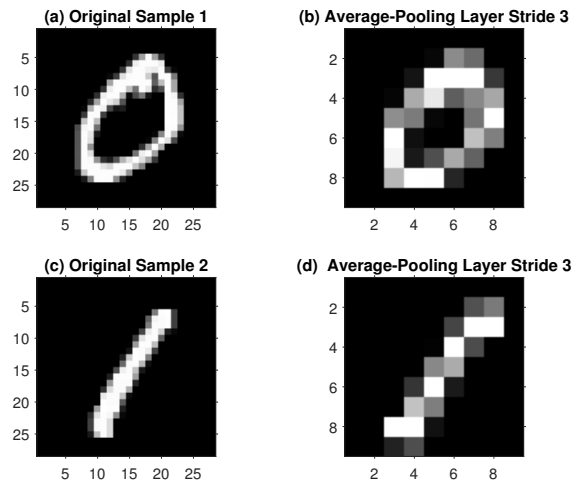
Other parameters that are required by some computational libraries are those used for regularization, initial weights initialization and which activation function will be used, details on these parameters were studied in (LECUN et al., 1990; GLOROT; BENGIO, 2010; HE et al., 2015).

Pooling Layers

In this part, pooling layers are introduced, which serve for mitigating the sensitivity of convolutional layers to location and of spatially downsampling representations.

Pooling, like convolutional layers, make use of fixed-shape windows that are slid over the input according to their stride. However, unlike convolutional layers, pooling layers use a single filter and, in general, do not have learnable parameters. Therefore, the size of the filter and the stride are the two parameters involved in pooling layers. Pooling operators

Figure 9 – Example of Average Pooling



Images (b) and (d) were obtained applying 3-stride average downsampling on images (a) and (c), respectively.

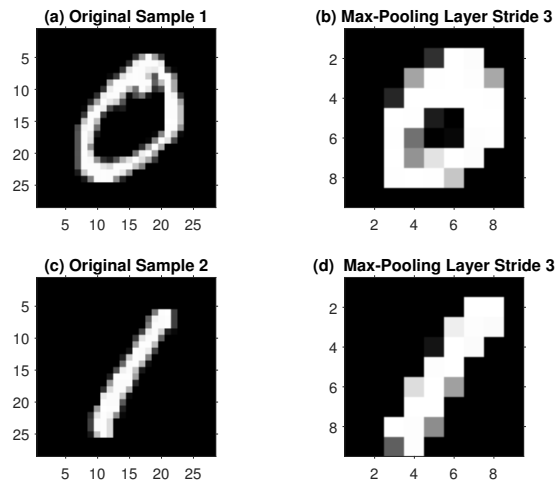
compute an operator such as minimum, median, average or maximum over each single slide of data. We will focus on max and average operators, which are the basic operations of the so called max-pooling and average pooling layers.

In general, it is not possible to determine which method is better than the other. The choice of the pooling operations is data dependent. On the one hand, average pooling performs a downsampling smoothing original images. For a pixel (i, j) , the average pooling back pass is computed summing the values of the forward function output in which (i, j) is involved and the resulting value of that sum is then divided by the size of the filter (MITTAL, 2018). Figure 9 illustrates the result of the forward function of an average pooling layer with stride of 3, for two different samples. On the other hand, a max-pooling layer is a sample-based discretization process that acts as router, allowing only those max values in the slides pass to the next layer. To compute the backward pass function in a max-pooling layer, it is common to keep track of the switch indices of max activation at the forward pass of a pooling layer so that gradient routing is efficient during BP. In other words, the values of gradients passed to the previous layer are zero except in those places where values of input have the greatest value at the forward computation. A complete description on max-pooling layers can be found in (NAGI et al., 2011; MITTAL, 2018).

Weighted Connected Layers

Weighted connected layers compute an output value aggregating inputs using a vector of weights and a bias. Specially, fully weighted connected layers are hidden layers that compute a sum of multiplication of some weights and inputs at each node. In contrast to convolution layers, the receptive field of fully connected layers is all the input volume.

Figure 10 – Example of Max Pooling



Images (b) and (d) were obtained applying 32-stride max-pooling downsampling on images (a) and (c), respectively.

Basically, fully weighted connected layers follow the same principle of traditional linear layers of multi-layer perceptrons neural network (MLPs).

The learnable parameters of a fully connected layer are a set of vectors of weights for each node and its respective bias values. In general, a layer represents the whole set of weights as a matrix, where each row represents an individual set of neural weights and a vector that represents a bias for each neuron. Hence, the forward pass computes:

$$F_{\ell}(\mathbf{x}) = W \cdot \mathbf{x} + \mathbf{b}, \quad (3.22)$$

where W is the matrix of associated weights, \mathbf{b} is a vector of bias values and \mathbf{x} is the input vector coming from the previous layer.

The backward pass simply computes the gradients of these functions taking into account the gradient vector received from the next layer. A standard is to assume an activation function after a fully connected layer.


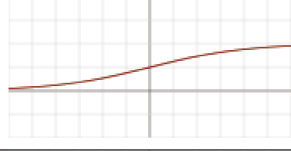
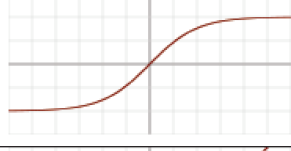
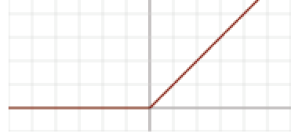
A comprehensive and complete description of fully connected layers, their initialization methods and regularization of parameters can be found in (GLOROT; BENGIO, 2010; GOODFELLOW; BENGIO; COURVILLE, 2016).

Activation Layers

Activation layers are sometimes considered hidden layers and an abstraction of the firing potential of a neuron function. It has been proven in MLP that non-linear functions play a fundamental role in the neural network processing. For example, a set of fully connected layer without any activation layer (or identity activation) is equivalent to a single-layer model. On the other hand, when the activation functions of an architecture

are non-linear such as sigmoidal functions, two layers of fully connected (followed by these activations) are enough to prove the universal approximation theorem (CYBENKO, 1989). An activation layer does not have any learnable parameter, Table 2 summarizes the most

Table 2 – Activation Functions .

Name	Equation	Derivative	Plot
Identity	$f(x) = x$	$f'(x) = 1$	
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	
hyperbolic tangent	$f(x) = \tanh(x)$	$f'(x) = 1 - f(x)^2$	
RELU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x > 0 \end{cases}$	

basic activation functions used in NN, its derivatives, and a sample plot in which the behavior of each function is shown. In most of the modes presented in this work, linear rectifiers appear frequently.

The reader may note that ReLu is not differentiable in some points. In this case, it does not represent a problem, a convention is to consider a zero value whenever a differentiation problem appears.

It is important to mention that some authors and frameworks do not consider the activation function as a layer itself. Instead, the activation function is a parameter of other layers, for example, a fully connected layer or a convolution layer. Therefore, some layers have activation functions attached to the computation of their own operations. Moreover, in the biological description of artificial neural networks activation functions are considered part of the neuron computation. Nevertheless, for implementation purposes and considering Equation 3.15 an activation function can be considered an independent layer.

Dropout Layers

One of the major concerns on DL is the overfitting problem. A model with a high numbers of parameters can fit to closely or even exactly a training set. Nevertheless, may fail to fit additional data at the test stage. The number of layers and parameters of a DL model depends much on the architecture. Drop Layers are a simple way to prevent

overfitting. The main idea of “dropout” is to drop out units at hidden and visible layers in a neural network. In that sense, at the training stage, dropout forward functions drop nodes or regions out from the network with a probability p so that some parameters are excluded. This strategy reduces interdependent learning and, therefore, smoothes overfitting. Here, we discuss two kind of dropouts, dropout (DO) and connected dropout (CD) strategies. On the one hand, in DO Layers a random fraction p of the input values is ignored (setting zero values). DO strategy was first presented in (SRIVASTAVA et al., 2014) and can be used after most types of layers, such as fully connected layers and convolutional layers. For example, FC layers use most of the input to compute its transformations. Therefore, nodes develop co-dependency among each other during training which leads to overfitting and, consequently, poor feature learning. Figure 11 illustrates the case of DO applied to FC layers.

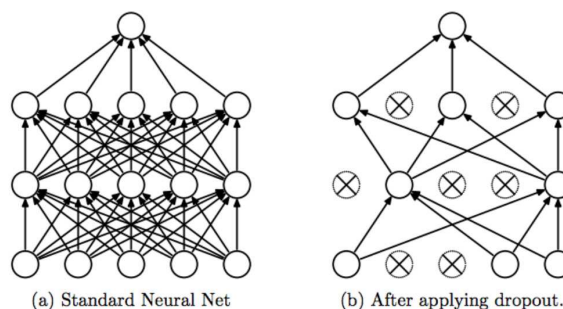
On the one hand, the case of DO applied to spatial structures, such as images, can be compared to the introduction of pepper noise to inputs and it is not as effective as in the FC case. Figure 12 illustrates the case of DO applied to digit images.

On the other hand, CD layers use spatial information and is applied to special structures, such as images and is only applied to those layers in which outputs have spatial information, such as convolutional or pooling layers. Instead of introducing pepper noise, CD layers introduce a kind of information occlusion per region. Hence, this layer ignores with probability p sub-regions of the structure. Figure 13 shows an image obtained after CD application on digit images.

Batch Normalization Layer

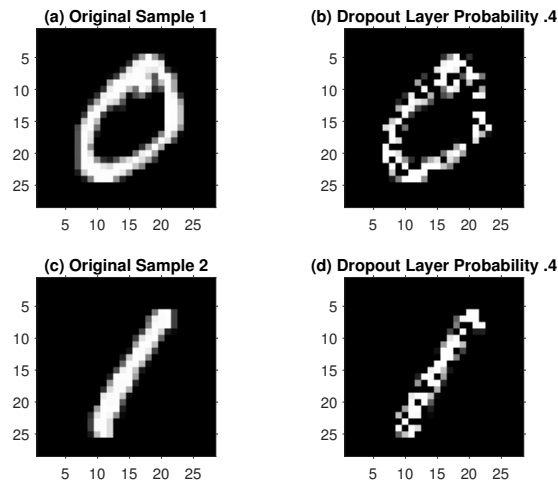
Batch normalization (BN) was developed to deal with the exploding and vanishing gradient problems, which cause gradient values of early layers to reduce or increase exponentially its magnitude. Besides, it helps to handle the internal covariance

Figure 11 – Example of Dropout in FC layers



Application of dropout operation with $p = .4$ on FC Layers. Source: passrivastava2014dropout

Figure 12 – Example of Dropout

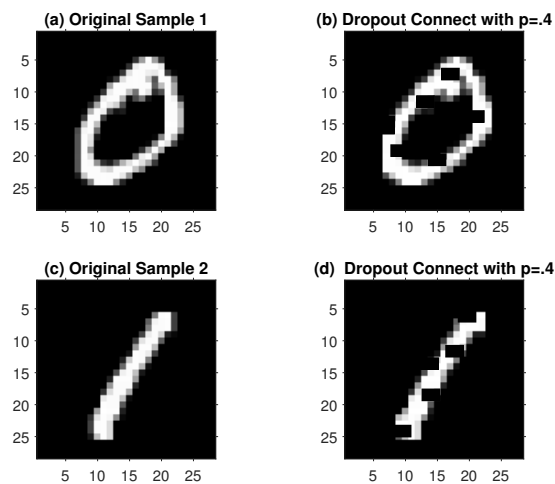


Figures (b) and (d) were obtained applying dropout in images with $p = .4$ on images (a) and (c), respectively.

shift, which is another issue that is common in backpropagation (IOFFE; SZEGEDY, 2015).

The internal covariance shift is related with changes of parameters during the training phase, especially when inputs at early hidden layers are constantly changing and therefore, inputs to later layers can become not stable causing a lower convergence during training. Batch normalization is intended to deal with this lack of stability. In addition, BN layer has shown reductions on the sensitivity to random network parameters initialization. Normalization layers are added between hidden layers and its neurons create new features

Figure 13 – Example of Dropout Connect



Images (b) and (d) were obtained applying dropout connect with $p = .4$ on images (a) and (c), respectively.

that regulate the covariance. It has additional parameters μ_j and σ_j which are intended to adjust the level of normalization of in the j -th unit. The parameters are trained by mini-batches. The proposal is to compute the mean μ_j and variance σ_j over each single mini-batch. A single node of a BN layer F_ℓ computes the normalized activation:

$$\hat{x}_i^j = \frac{x_i^j - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad (3.23)$$

where ϵ is a very small positive value that improves numerical stability, x_i^j is the i -th previous layer output of the j -th element of a batch and μ, σ are computed for a mini-batch S with m instances as follows:

$$\mu_i = \frac{\sum_{j=1}^m x_i^j}{m} \quad \forall i \quad (3.24)$$

$$\sigma_i^2 = \frac{\sum_{j=1}^m (x_i^j - \mu_i)^2}{m} \quad \forall i \quad (3.25)$$

Finally, outputs are given shifting and scaling these activations, that is, for an element $z^j \in \mathcal{S}$ that belong to the mini-batch being processed, $x^j = G_{\ell-1}(z^j)$:

$$f_i^\ell(x^j) = \gamma_i \cdot \hat{x}_i^j + \beta_i \quad \forall i, j \quad (3.26)$$

Here, the offset β and the scale factor γ are learnable parameters. In general, these types of layers perform better when applied before the activation functions (i.e., before the application of a ReLU function and after the transformation.)

After training, the batch normalization layer recalculates the mean and the variance, taking into account the whole dataset. When the network is used to make predictions on new data it uses these new mean and variances to normalize the activations. The activations of this normalization will depend on the relations between each example in the mini-batch. We need to compute the backward pass function and gradients of the learnable parameters γ and β , these derivatives are given by:

$$\frac{\partial \mathcal{L}}{\partial \beta_i} = \sum_{j=1}^m \frac{\partial L}{\partial f_i^\ell(x^j)} \quad (3.27)$$

$$\frac{\partial L}{\partial \gamma_i} = \sum_{j=1}^m \frac{\partial L}{\partial f_i^\ell(x^j)} \cdot \hat{x}_i^j \quad (3.28)$$

In terms of x_i^j , we have the following

$$\frac{\partial \mathcal{L}}{\partial x_i^j} = \frac{\partial \mathcal{L}}{\partial \hat{x}_i^j} \cdot \left(\frac{1}{\sigma_i}\right) + \frac{\partial \mathcal{L}}{\partial \mu_i} \cdot \left(\frac{1}{m}\right) + \frac{\partial \mathcal{L}}{\partial \sigma_i^2} \cdot \left(2 \cdot \frac{x_i^j - \mu_i}{m}\right) \quad (3.29)$$

where:

$$\frac{\mathcal{L}}{\sigma_i^2} = -\frac{1}{\sigma_i^3} \cdot \sum_{j=1}^m \frac{\partial \mathcal{L}}{\partial f_i^\ell(x^j)} \cdot \gamma_i \cdot (x_i^j - \mu_i) \quad (3.30)$$

$$\frac{\partial \mathcal{L}}{\partial \mu_i} = -\frac{\gamma_i}{\sigma_i} \sum_{j=1}^m \frac{\mathcal{L}}{\partial \hat{x}_i^j} + \frac{1}{3} \cdot \left(\sum_{j=1}^m \frac{\partial \mathcal{L}}{\partial f_i^\ell(x^j)} \gamma_i \cdot (x_i^j - \mu_i) \right) \cdot \left(\frac{\sum_{j=1}^m (x_i^j - \mu_i)}{m} \right) \quad (3.31)$$

BN layers also act as weight regularizers. Since individual points are used in terms of the mini-batch, an observation x^j could cause different updates depending on which batch it is included in. In general, BN is used between fully weighted connected layers and activation layers.

The Softmax Layer Case

The function softmax is computed with respect to multiple inputs and therefore, is a special kind of activation function. In addition, softmax function is considered as an output layer. In general, attached with the cross entropy loss function given by Equation 3.5.

The softmax layer F_ℓ converts $k = |F_{\ell-1}|$ real-valued predictions coming from the previous layer $v_i = f_i^{\ell-1}(\mathbf{x})$, using the following formula:

$$o_i = f_i^\ell(v_i) = \frac{e^{v_i}}{\sum_{j=1}^k e^{v_j}} \quad \forall i = 1, \dots, k \quad (3.32)$$

A first insight is that if we try to use the chain rule to apply backpropagation then we have to compute partial derivatives with respect to each input and output. However, when we take into account the cross entropy loss and the softmax as the output layer, the equation becomes:

$$\mathcal{L} = - \sum_{j=1}^k y_j \cdot \log(o_j) \quad (3.33)$$

then the derivatives with respect to the layer $F_{\ell-1}$, $\frac{\partial L}{\partial v_i}$ have a particularly simple form:

$$\frac{\partial L}{\partial v_i} = \sum_{j=1}^k \frac{\partial L}{\partial o_j} \cdot \frac{\partial o_j}{\partial v_i} = o_i - y_i \quad (3.34)$$

In the case of architectures that use backpropagation with a coupled cross entropy loss function, the error propagation goes first from the output layer to v_1, \dots, v_k , then the backpropagation proceed normally, following the structure of earlier layers. Softmax layers are discussed in (GOODFELLOW; BENGIO; COURVILLE, 2016).

3.5 Some Deep Learning Architectures

It is not possible to get functional architectures stacking layers in a random way and obviously the layer sequences must have a certain logic, recalling the objective is to progressively extract features from raw inputs.

In this section, we present two functional and popular deep learning architectures that make use of these previously studied layers.

Stacked Autoencoders Based Neural Network

An autoencoder was developed to learn data coding and to automatically extract features from raw data in a unsupervised fashion (VINCENT et al., 2010). In general, an autoencoder performs a reduction of dimensionality, training a hidden layer with a reduced number of nodes, both functions are fully weighted connected layers, possibly followed by an activation function. The idea is to generate an encoding as close as possible to its original input. In particular, sparse regularized autoencoders apply an L_1 regularization on the weights (MAKHZANI; FREY, 2013; ARPIT et al., 2015). Autoencoders are effective for feature extraction. The extracted features have been widely used for subsequent classification.

Formally, a single autoencoder is composed of two parts, an encoder $h : \mathbb{R}^n \rightarrow \mathbb{R}^m$ that maps inputs to compressed representation and a decoder $r : \mathbb{R}^m \rightarrow \mathbb{R}^n$ which maps encoded representation into its original space (HINTON; SALAKHUTDINOV, 2006).

The idea to use these functions is simple: Given a set of unlabeled observations we would like to learn the encoder and decoder functions h and r such that $g(x) = r(h(x))$ becomes an approximation of the identity function. In words, an autoencoder is trained to produce as output its own inputs such that the reconstruction error is minimized and it can be trained by SGD, replacing the desired labels and putting output equal to input values.

Stacked autoencoders train a set of encoder in an organized manner. The idea is to train a set of autoencoders $g_i, i = 1, \dots, k$ progressively and then discard the decoder and keep the encoding part h . The first stage is to train an encoder g_1 using all data observation in the training dataset and keep the function h_1 . Next, an autoencoder g_2 is trained using as input-output the transformations produced by the encoder h_1 . In this sense, an autoencoder g_j can be trained using the representation encoded by h_{j-1} . The last transformation h_k is trained using a softmax layer, this time using the desired labels as an output. Finally, a gradient descent runs over the whole stacked net which produces the desired network. A single autoencoder and an example of stacked encoders are shown in Figure 14. In that example, a set of encoders refines the data at each level and softmax with a CE loss function perform the classification.

Convolutional Neural Network

Convolutional neural networks (CNN) have been widely discussed and there are many articles, books and international publications on the subject. However, in this part, we will introduce the idea behind CNN. In addition, an example of image decomposition performed by CNN is presented.

CNNs are most commonly applied to image analysis and classification. Instead of the fully weighted connected layers used in multilayer perceptron, a CNN makes use of the previously discussed convolutional layers.

Figure 14 – Autoencoders

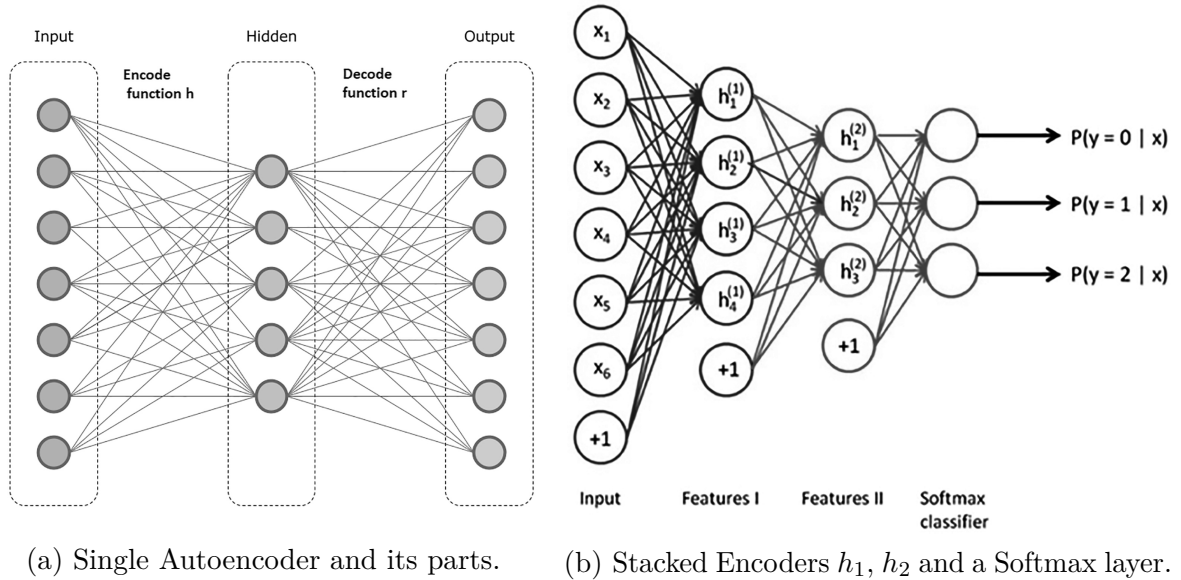
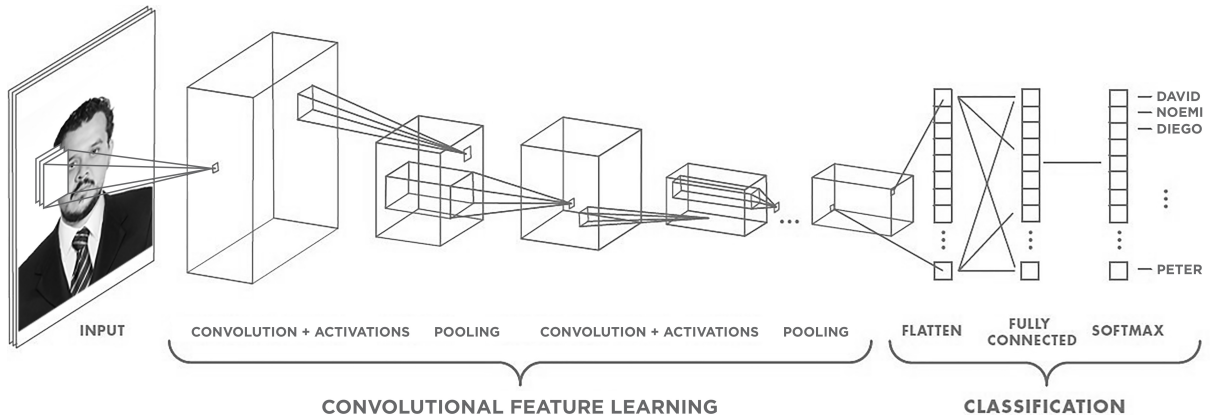


Table 3 – Reduced Notations to Describe Layers.

	Representation	Parameters
Fully Connected Layer	FC n	n: Represents the number of nodes.
Sliding Convolution Layer	SC n f s p	n: Represents the number of filters. f: Represents the size of the filters. s: Represents the size of the stride. p: If the layer has not zero padding.
Max Pooling Layer	PM s	s: Represents the size of the stride.
Dropout Layer	DO p	p: Dropout probability.
ReLU Activation Layer	R	none
Batch Normalization Layer	BN	none
Softmax Layer	SM	none

Figure 15 – Basic CNN architecture



An abstract example of a CNN that illustrates the convolutional and classification part of the architecture.

CNN were inspired by the connectivity between biological neurons of the animal visual cortex. Each neuron on the cortical area responds to a stimulus of a restricted region of the visual field, which is named the receptive field. In order to simulate this behaviour, CNNs make use of a process that involves filters over certain regions of input data.

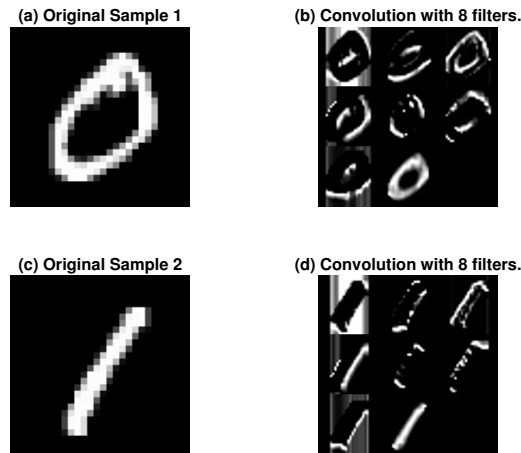
Broadly speaking, CNNs have an input, an output layer and a set of hidden layers. A basic CNN structure consists of a convolutional layer, followed by an activation and a pooling layer. The output can be used as an input to repeat the process again. We refer to these layers as the convolutional part of the model. Finally, a classifier system, or classification part of the model gives the final output. For example, two fully connected layers, where in the second layer each node represents a class, is a common choice for the classification part. The softmax layer together with the cross entropy loss is one of the most used cost functions to apply backpropagation. Figure 15 shows a visual representation of the process and the involved parts.

There are many CNN architectures with different compositions, number of layers and configurations. Let us introduce some notations to describe deep learning architectures used throughout this work. Table 3 summarizes short notations of individual layers reviewed in last section. Next, we present an example of an architecture and its description using these short notations.

Example 5. Consider a convolutional neural network F with seven layers that is set to perform classification on grayscale images with size 28×28 and 10 classes and let the configuration of the network as follows:

1. An input layer that performs an identity function over an image 28×28 denoted by a matrix I that computes an identity function.
2. A convolution layer with 8 kernels, all 5×5 sized, with stride 1 and zero padding,

Figure 16 – Example of CNN Processing - First layers.

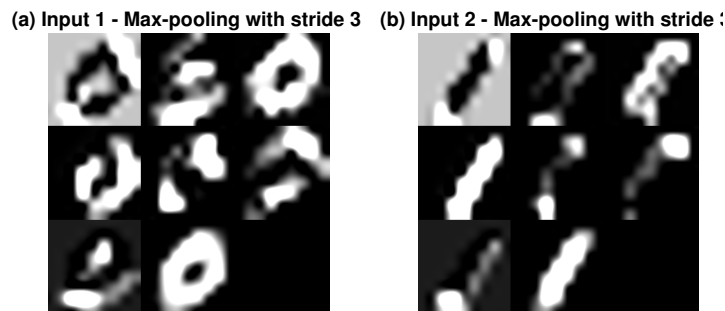


(a) and (c) are two original digits images used as an input for the network. (b) and (d) represent, respectively, the transformation of these inputs by the $SC16|3|1 + R$ with 8 filters.

denoted by $SC6|8|1$. Assume a $ReLU$ activation function R . Figure 16 shows an example of the transformation performed by the composition of these first three layers.

3. A max-pooling layer F_3 that performs downsampling operation using a 3×3 sized window denoted by $MP|3$. An example of how this layer process samples is illustrated in Figure 17. The input of this example is given by the output shown in Figure

Figure 17 – Example of CNN Processing - Max-pooling layer.

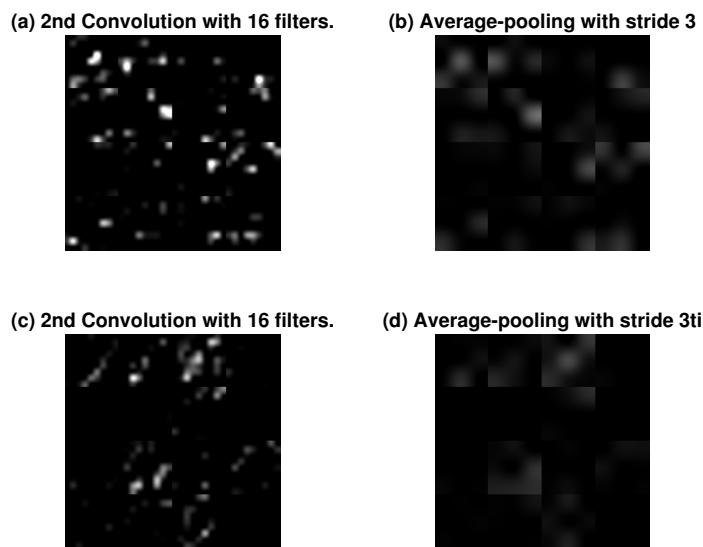


Transformation performed by a max-pooling. We consider as inputs, the output shown in Figure 16

4. A second convolutional layer with 16 kernels all 5×5 sized, with zero padding and stride of 1 denoted by $SC16|5|1$. A $ReLU$ activation function is assumed.

5. A second pooling layer F_5 that computes average downsampling and uses a 3×3 sized window, denoted $PA/4$. Figure 18 illustrates an example of the processing performed by the second convolution layer and the average layer.
6. A fully connected layer with 10 nodes, each node representing a class is denoted $FC10$.
7. The loss function used is given by the MSE given in Equation 3.3.

Figure 18 – Example of CNN Processing - Second Convolution and Average Layer.



Transformation performed by the layers described at 4, 5, 6 of the example. We consider as inputs, the output shown in Figure 17

In this example, these layers described from point 1 to 6 constitute the convolutional part of the network. On the other hand, layers described in points 7 and 8 work as the classification part.

In general, the last example shows in a visual way the process of forward decomposition of an image computed by convolutional neural networks. Observe that there are many applications of DL, the case of CNNs discussed in this section is a special case widely used for classification and it is an important case of study in this thesis. Other architectures and applications are reviewed in (SCHMIDHUBER, 2015; HE et al., 2016). Parameters initialization and activation functions are discussed in (GLOROT; BENGIO, 2010; HE et al., 2015). A complete and well organized book that contains a detailed discussion of DL was published by Goodfellow et al., (GOODFELLOW; BENGIO; COURVILLE, 2016).

4 A Review of Morphological Perceptrons and Related Models

There must be a trick to the train of thought, a recursive formula. A group of neurons starts working automatically, sometimes without external impulse. It is a kind of iterative process with a growing pattern. It wanders about in the brain, and the way it happens must depend on the memory of similar patterns.

Stanislaw M. Ulam, *Adventures of a Mathematician*

In this chapter, we review the morphological perceptrons, some related models and its geometrical interpretations. We also discuss some algorithms to train these architectures and the framework in which MPs have been developed.

The MP/CL idea is to decompose functions in terms of elementary MM operators. This idea implicitly lies in the core of several MM training algorithms and models (SUSSNER; ESMI, 2009b; SUSSNER; ESMI, 2011). A special emphasis was given to classification problems in lattices \mathbb{L}^n where the structure \mathbb{L} is an ℓ -group. The competitive layer is a maxout layer responsible for selecting the hyperbox that best approximates each target. Despite these advances, the number of hyperboxes determined by Sussner's and Esmi's algorithm tends to grow very quickly and as a consequence, becomes subject to overfitting. In particular, the MP/CL becomes subject to overfitting when applied to sparse, high-dimensional data (SUSSNER; CAMPIOTTI, 2020). Nevertheless, among MPs, MP/CL became a standard model and since then, new and more sophisticated MP algorithms and learning techniques for calculating the hyperboxes that separate the classes have appeared. This field is maturing, with a wealth of well understood methods and algorithms to train MPs. Section 4.4 presents a brief review of some algorithms and architectures from literature. Additionally, Section 4.5, summarizes hybrid models in which morphological neurons are combined with other kinds of neurons.

As a result of Theorem 16, in order to define the four basic operations of MM in grayscale, it is enough to define an algebraic erosion, dilation and negation. Note that a digital

grayscale image can be viewed as a function $X \rightarrow \mathbb{G}$, where \mathbb{G} denotes the set of extended real numbers $\mathbb{R}_{\pm\infty}$ or the extended integer numbers $\mathbb{Z}_{\pm\infty}$ and X is the set of pixels that is usually given by a subset of \mathbb{R}^d or \mathbb{Z}^d . A more general approach would be to consider \mathbb{G} a complete extension of an ℓ -group.

Observe that the basic computations in an MNN were first proposed on the complete ℓ -group extensions. A more general approach would be to consider a bounded ℓ -group extensions \mathbb{E} . Observe that, the basic computations of MNN occurring in a morphological network were first proposed on the algebraic complete lattice ordered group structures: $\mathbb{R}_{\pm\infty}$ and $\mathbb{Z}_{\pm\infty}$. It was aforementioned that the operations $+$ and $+$ ' only differ from each other whenever the bounds ∞ and $-\infty$ are involved, as determined in Equations 2.34 and 2.35 and for any $x \in \mathbb{R}$ the operations $+$ and $+$ ' behave in the usual way: $(+\infty) + x = x + (+\infty) = +\infty$ and $(-\infty) + x = x + (-\infty) = -\infty$ (See Section 2.4).

Let $\mathbf{a} \in \mathbb{G}^X$ be an image and $\mathbf{s} : X \rightarrow \mathbb{G}^X$ a structuring element and the operators $D(\mathbf{a}, \mathbf{s}), E(\mathbf{a}, \mathbf{s}) : X \rightarrow \mathbb{G}$ defined by:

$$D(\mathbf{a}, \mathbf{s})(\mathbf{x}) = \bigvee_{\mathbf{y} \in X} (\mathbf{a}(\mathbf{y}) + (\mathbf{s}_{\mathbf{y}}(\mathbf{x}))) \quad \text{and} \quad E(\mathbf{a}, \mathbf{s})(\mathbf{x}) = \bigwedge_{\mathbf{y} \in X} (\mathbf{a}(\mathbf{y}) +' (\mathbf{s}_{\mathbf{x}}(\mathbf{y}))^*) \quad (4.1)$$

Equations in 4.1 can be written in terms of max-product and min-product, whenever the set $|X|=n$ is finite. Therefore, following Theorem 17, D and E are a dilation and an erosion, respectively. Note also that the structuring element \mathbf{s} is varying in X .

MM is a theory focused on shape analysis for analysis of planar and spatial structures, which is based on set theory, algebra and geometry. Although MM was developed for digital image analysis, it can be employed in other object graphs, surface meshes, solids, and many other spatial structures. However, the main idea of MM analysis is to extract information from relationships between an image and a probe (named structuring element), which is a predefined small shape that can be fixed or variable, which is the case of Equation 4.1. In the case of images, morphological operators measure how a structuring element fits local neighborhoods of the pixels. The morphological erosion of an image by some structuring elements at certain point \mathbf{x} , is computed by measuring how the translated structuring elements fits at that point. For example, the fuzzy morphological erosion of an image \mathbf{a} by some structuring element \mathbf{s} at a certain point \mathbf{x} yields a degree of inclusion of subsethood of a translated version of \mathbf{s} in \mathbf{a} .

Examples of MNN that perform algebraic dilations and erosions are: Grayscale Morphological associative memories (SUSSNER, 2001; SUSSNER, 2003b; SUSSNER, 2003a; VALLE; SUSSNER; GOMIDE, 2004), fuzzy morphological associative memories which execute basic operations of fuzzy mathematical morphology in the recall phase (SUSSNER; VALLE, 2006b; VALLE; SUSSNER, 2008b; SUSSNER; VALLE, 2006a; VALLE; SUSSNER, 2008a). Examples of MNNs that perform morphological operators of input vectors by weight vectors or vice versa include: Kosko subsethood-FAMs (SUSSNER et al., 2012), θ -FAMs (ESMI et al., 2014), fuzzy lattice reasoning models of Kaburlassos (PETRIDIS;

KABURLASOS, 1998; KABURLASOS; PETRIDIS, 2000; KABURLASOS; KEHAGIAS, 2013) and, recently submethod interval associative memories θ -FAMs (SUSSNER; ESMI; JARDIM, 2019).

Definition 35. *Morphological neural networks* are a subclass of ANNs that perform morphological operators at each node. Particularly, **Morphological Perceptrons** are part of the MNNs and are strongly tied to supervised learning and regression. In a general way, MPs can be defined as feedforward MNNs with applications on classification or regression.

MPs were first described in the mid 1990s in (SUSSNER, 1998). A learning rule and a training algorithm for binary classification were given in (SUSSNER, 1998). Minimax algebra allows an easy way to define real-valued parameters in MNNs. Grayscale MM can be embedded into minimax algebra and is the framework in which MNNs are described (DAVIDSON; HUMMER, 1993). Hence, it is not unexpected that several models of MNNs employ the minimax algebra defined in (CUNINGHAME-GREEN, 1991). In this chapter, we briefly review some models based on mathematical morphology. In Section 4.1 is given a brief review of the first learning rule and algorithm to train MPs (SUSSNER, 1998). In Section 4.2 the morphological perceptron with competitive learning and the framework introduced by Sussner and Esmi (SUSSNER; ESMI, 2009b) are reviewed. Moreover, in this section some relationships between MPs and concepts studied in Section 2.5 are deepened. In Section 4.3, the dendrite morphological neural network presented in (RITTER; IANCU; URCID, 2003) is reviewed. In section 4.4 some architectures and algorithms related with MPs are presented. Section 4.5 summarizes some of the relevant hybrid and morphological related models developed to perform supervised learning tasks.

4.1 The Original Morphological Perceptron Model

The first architecture and approach to train a morphological neuron were discussed in (RITTER; SUSSNER, 1996). The first supervised learning algorithm for multilayer morphological perceptrons was introduced in (SUSSNER, 1998). The supervised learning algorithm makes use of a hard-limiting activation function.

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \quad (4.2)$$

Let $W = \{w_1, \dots, w_n\}$, where $w_i \in \mathbb{R}$, $x \in \mathbb{R}^n$ and a bias $b \in \mathbb{R}$ and let y^0 and y^1 be two labels, then the difference between the computation of MP and linear perceptron for binary classification is:

1. A linear perceptron assigns a pattern x to a class y^0 if:

$$f\left(\left[\sum_{i=1}^n x_j \cdot w_j\right] - b\right) = 0 \quad (4.3)$$

and assign a class y^1 , otherwise.

2. A morphological perceptron assign a pattern x to a class y^0 if:

$$f\left(\left[\bigvee_{i=1}^n x_j + w_j\right] - b\right) = 0, \quad (4.4)$$

otherwise the assigned class is y^1 .

A dual model in which a pattern x belongs to a class y^1 whenever:

$$f\left(\left[\bigwedge_{i=1}^n x_j + w_j\right] - b\right) = 1 \quad (4.5)$$

was also introduced. Since this duality between these two equations, the results and argumentation of models involving 4.4 and 4.5 are similar.

It was also argued that the bias parameter was not required in the computation, it is:

$$\left(\bigvee_{j=1}^n x_j + w_j\right) - b = \bigvee_{j=1}^n x_j + (w_j - b) \quad (4.6)$$

and therefore weights $W' = (w_1 + b, \dots, w_n + b)$ with $b' = 0$ do not change the behavior of models. However, the surfaces that can be represented with these models are very limited. To increase the power of representation a set of parameters $a_j \in \{-1, 1\}$, $j = 1, \dots, n$ was introduced, the resulting model was called generalized single layer morphological perceptron. The rule associated to this model assigns the label y_0 to a pattern $x \in \mathbb{R}^n$ whenever

$$f\left(\bigvee_{i=1}^n a_j(x_j + w_j)\right) = 0 \quad (4.7)$$

where the parameter a_j represent the j -th pre-synaptic response. If $a_j = 1$ the response is said to be excitatory and if $a_j = -1$ the pre-synaptic response is said to be inhibitory. Just like the linear perceptron, an evident limitation of these model appears when considering the XOR problem. The feedforward two-layer morphological perceptron was presented to deal with that problem. The correctness of the algorithm and solutions to solve XOR, AND, OR problem were also proven (RITTER; SUSSNER, 1996). Moreover, solutions to the generalized n-dimensional AND/OR problems and an algorithm that takes a training set and determines the number of hidden nodes necessary to perform binary separation were presented.

Sussner introduced a first supervised learning algorithm for a multi-layer morphological

perceptron (SUSSNER, 1998). In this version, a set of hyperboxes acts as parameters $[\underline{\mathbf{w}}^j, \overline{\mathbf{w}}^j]$ for $j = 1, \dots, m$, this is:

$$g(\mathbf{x}) = f\left(\left(\bigwedge_{i=1} (x_i +' (\underline{w}_i^1)^*) \wedge \bigwedge_{i=1} (\overline{w}_i^1 +' (x_i)^*)\right) \vee \dots \vee \left(\bigwedge_{i=1} (x_i +' (\underline{w}_i^m)^*) \wedge \bigwedge_{i=1} (\overline{w}_i^m +' (x_i)^*)\right)\right) \quad (4.8)$$

Remark that a pattern $\mathbf{x} \in \mathbb{R}^n$ belong to a class y^0 , whenever $g(x) = 1$. So that the goal of an algorithm is to encapsulate all patterns that belong to a certain class in hyperboxes. Although a first functional algorithm to train MPs was presented, it only works for binary classification.

It was already mentioned that MPs are feedforward MNNs designed to perform classification or regression tasks. In this sense, Equation 4.8 can be written in terms of min-product operators pairs as follows:

$$g(\mathbf{x}) = f\left(\bigwedge_{j=1}^m ((\mathbf{w}^j)^* \boxtimes \mathbf{x}) \wedge (\mathbf{x}^* \boxtimes \mathbf{w}^j)\right), \quad (4.9)$$

where \mathbf{w}^* denotes the conjugate of the vector \mathbf{w} given in Definition 21 and \boxtimes, \boxtimes denote max-product, min-product operators between two vectors given in Definition 26.

Equation 4.9 describes MPs in terms of minimax operators and makes clear the relationship between minimax algebra and MPs. As aforementioned Equation 4.1 can be written in terms of min-product and max-product, whenever \mathbf{x}, \mathbf{w} are dimensional finite, which is the case of Equation 4.9.

Given the complete ℓ -group extension $\mathbb{R}_{\pm\infty}^n$, an input $\mathbf{x} \in \mathbb{R}_{\pm\infty}^n$ (in practice, datasets are usually defined in \mathbb{R}^n). If we consider Equations 4.1, synaptic weights are vector $\mathbf{w} \in (\mathbb{R}^n)_{\pm\infty}$ and neurons in a hidden layer compute an output y according to one of the following rules:

$$y = f(\varepsilon_{\mathbf{w}}(\mathbf{x})) \text{ where } \varepsilon_{\mathbf{w}}(\mathbf{x}) = \bigwedge_{i=1}^n (x_i +' w_i) \quad (4.10)$$

$$y = f(\delta_{\mathbf{w}}(\mathbf{x})) \text{ where } \delta_{\mathbf{w}}(\mathbf{x}) = \bigvee_{i=1}^n (x_i + w_i) \quad (4.11)$$

$$y = f(\overline{\varepsilon}_{\mathbf{w}}(\mathbf{x})) \text{ where } \overline{\varepsilon}_{\mathbf{w}}(\mathbf{x}) = \bigvee_{i=1}^n (x_i^* + w_i) \quad (4.12)$$

$$y = f(\overline{\delta}_{\mathbf{w}}(\mathbf{x})) \text{ where } \overline{\delta}_{\mathbf{w}}(\mathbf{x}) = \bigwedge_{i=1}^n (x_i^* +' w_i) \quad (4.13)$$

which are given by the max and min-product given in Definition 26. The units of computation of MPs were studied in terms of basic MM operators (SUSSNER; ESMI, 2009b). Since mappings between complete lattice can be expressed in terms of the infimum of erosion and anti-dilation or in terms of the supremum of dilation and anti-erosion operations.

Sussner and Esmi proposed to perform the unit of computation of MPs using anti-dilation and erosion operators (SUSSNER; ESMI, 2011), based on the aforementioned results of (BANON; BARRERA, 1993).

Let us consider the operation:

$$g(\mathbf{x}) = \varepsilon_{\mathbf{a}^*}(\mathbf{x}) \wedge \bar{\delta}_{\mathbf{b}}(\mathbf{x}) = \bigwedge_{i=1}^n (x_i +' a_i^*) \wedge (x_i^* +' b_i) \quad (4.14)$$

where $\mathbf{a} \leq \mathbf{b}$, ε is an erosion and $\bar{\delta}$ is an anti-dilation. Using Theorem 18, we can assure that $\mathbf{x} \in \mathbb{R}^n$ is contained in the interval or hyperbox $[\mathbf{a}, \mathbf{b}]$ if and only if $g(\mathbf{x}) \geq 0$. Moreover, note that Equation 4.14 can be computed in terms of two min-product operations. In fact, these products are morphological operators in the geometrical way in which the structuring element is variable. Moreover, this pairwise infimum of an erosion and anti-dilation can be accomplished in a single operation using an erosion, in this case, the input is a concatenation of \mathbf{x} , \mathbf{x}^* and the weight vector is given by concatenating \mathbf{a} and \mathbf{b} . Furthermore, Equation 4.14 can be expressed in terms of \mathbb{I} -set descriptors (Definition 15) and aggregators (Definition 27):

$$\begin{aligned} \varepsilon_{\mathbf{a}^*}(\mathbf{x}) \wedge (\bar{\delta})_{\mathbf{b}}(\mathbf{x}) &= \bigwedge_{i=1}^n (x_i +' (a_i)^*) \wedge \bigwedge_{i=1}^n (b_i +' (x_i)^*) \\ &= \bigwedge_{i=1}^n (x_i +' (a_i)^*) \wedge (b_i +' (x_i)^*) \\ &= \bigwedge_{i=1}^n \phi_{\wedge}(x_i, [a_i, b_i]) \end{aligned} \quad (4.15)$$

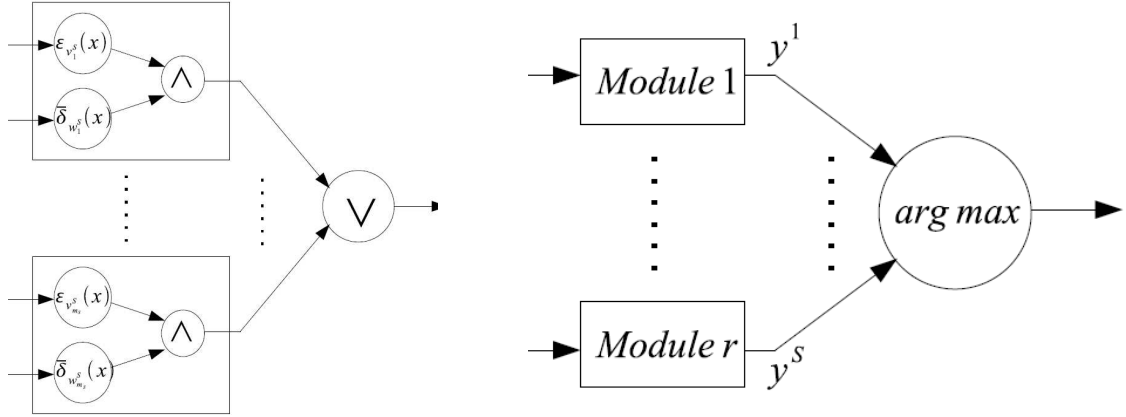
which is equivalent to the Equation 2.66. Note that ℓ -group extensions $\mathbb{E} = \mathbb{R}_{\pm\infty}$ and $\mathbb{E} = \mathbb{Z}_{\pm\infty}$ are extensions of Archimedean \circ -groups \mathbb{R} and \mathbb{Z} .

4.2 Morphological Perceptron with Competitive Learning

In morphological perceptrons with competitive layers (MP/CL), the hard limiter function f in equation 4.2 is replaced by an identity function and an argmax competition is carried out between the neurons in the output layer.

An approach to develop multiclass classification algorithms to train MPs is to extend binary classification training algorithms. Formally, let $\mathcal{T} = \{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^k, y^k)\}$ where $y_i \in \mathcal{L}$ and $\mathcal{L} = \{c^0, \dots, c^{n_c}\}$ is the label dataset for training with n_c classes, and let \mathcal{T}^s be the set of patterns \mathbf{x} associated with a class s for $s \in \mathcal{L}$.

For each $s = 1, \dots, n_c$ a binary classification can be applied with $\hat{C}^1 = \{(x, y) \in \mathcal{T} : y \in \mathcal{T}^s\}$, the set of patterns that belong to the class s and $\hat{C}^0 = \{(x, y) \in \mathcal{T} : y \notin \mathcal{T}^s\}$, the complement of \hat{C}^1 . This way the extended algorithm creates a set of sub-modules (hyperboxes) for each class $s = 1, \dots, n_c$ and the output of each module is given by a competition between sub-modules. We will refer to this process as the one vs all classes

Figure 19 – MP/CL s -th class module.

(a) A graphical representation of a class module. (b) Maxnet competition between modules.

Source: (SUSSNER; ESMI, 2009a)

strategy. Lastly, the number of output neurons is equal to the number of labels in the argmax competition stage. A graphical representation of this idea is illustrated in Figure 19a.

The output of a class-module s -th module is given by:

$$y^s = \bigvee_{j=1}^{m_s} (\varepsilon_{v_j^s}(\mathbf{x}) \wedge \bar{\delta}_{w_j^s}(\mathbf{x})) \quad (4.16)$$

where m_s is the number of hyperboxes of the s -th module. Observe that, the difference between Equations 4.8 and 4.16 is the absence of activation function in the Equation 4.16. However, it was mentioned that both equations are equivalent if we consider an identity function as activation.

The parameters \mathbf{w}_j^s and \mathbf{v}_j^s are the weights of the j -th submodule of the s -th module. Finally, an argmax competitions is done between all the modules. A winner takes all competition is performed through an argmax function, which attributes the winner label j to an input pattern. The Figure 19b shows how the competition between modules is performed. Figure 19 illustrates the competition between class modules.

4.3 Dendrite Morphological Neural Networks

In the early 2000s, Ritter et al., introduced a biological interpretation of MNNs in terms of neural dendrites (RITTER; IANCU; URCID, 2003). In dendrite morphological neural networks (DMNNs), the dendrite d^j of a neuron computes the output

upon presentation of an input \mathbf{x} using either one of the following equations:

$$d^j(\mathbf{x}) = p_j \bigvee_{i=1}^n a_{i,j}(x_i + w_i) \quad (4.17)$$

$$d^j(\mathbf{x}) = p_j \bigwedge_{i=1}^n a_{i,j}(x_i + w_i) \quad (4.18)$$

where $a_{i,j} \in \{-1, 1\}$ denote synaptic inhibition or excitation on the j -th neuron caused by the i -th input and $p_j \in \{-1, 1\}$ denotes the post synaptic output response (excitatory or inhibitory) of the j -th neuron. In this perspective, the unit of computation of neural networks are dendrites. We may observe that the contribution done in these models considered in Equations 4.17 and 4.18 is the introduction of synaptic output responses p_j . From this perspective, the argument to view dendrites as unit of computation is that the number of neuronal synapses is higher on the dendritic tree of the neuron and therefore, there is where the information is processed. This argument is not new and it has been widely studied in (SEGEV; RALL, 1988; MEL, 1993; KOCH; SEGEV et al., 1998; MEL, 1999). Besides, dendrites make up the largest component in both surface area and volume of the brain. Thus, dendrites cannot be ignored when attempting to model artificial brain networks, especially when some research have proposed that dendrites and not the neuron itself are the unit of computation of the brain.

The mechanism proposed in (RITTER; SUSSNER, 1996) makes use of lattice framework and replace the operation performed by a traditional neural network with lattice based operations.

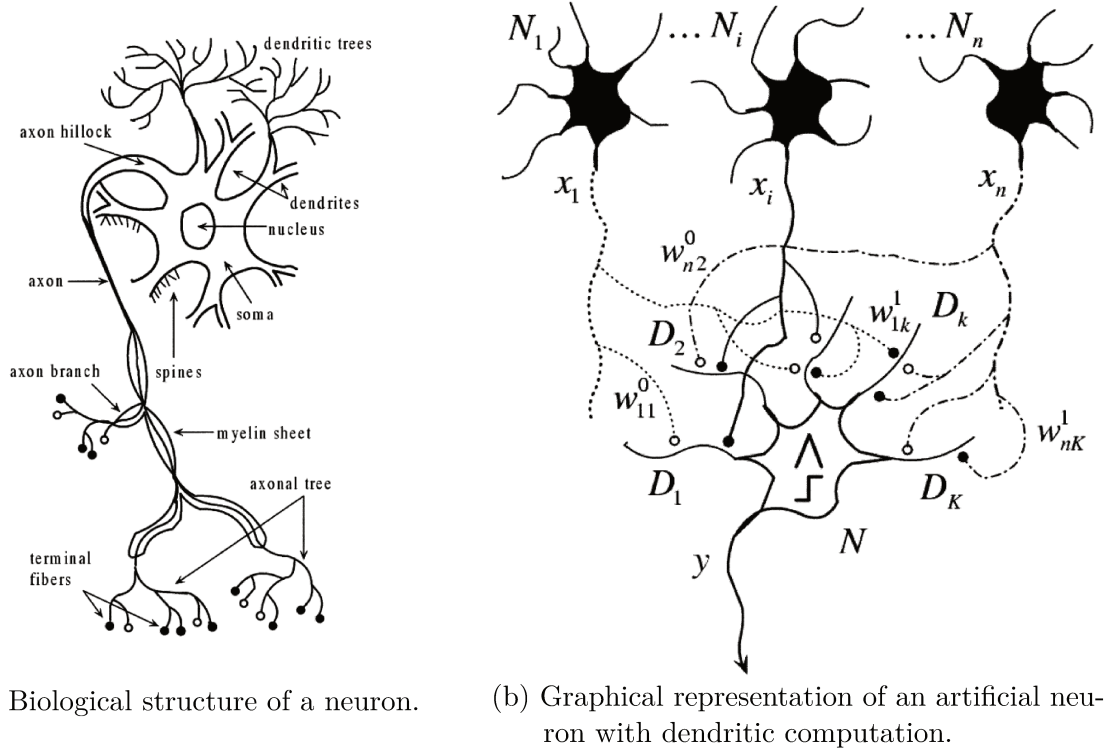
The single layer morphological perceptron is described as a single layer and feedforward neural network with n input neurons and m output neurons (RITTER; IANCU, 2003). At the output layer, neurons have dendritic structures that perform morphological operators based on lattice algebra. The axonal branch of neurons N_i for $i = 1, \dots, n$ makes synapses connection with dendrites of output neuron M_j , $j = 1, \dots, m$. In this model, the weight of a connection between N_i and the k -th dendrite of an output neuron M_j is denoted by $\mathbf{w}_{i,j,k}^\ell$, where the superscript $\ell \in \{0, 1\}$ distinguish between excitatory ($\ell = 1$) and inhibitory ($\ell = 0$) input to dendrite. The k -th dendrite of a j -th output neuron processes its inputs $\mathbf{x} \in \mathbb{R}^n$ and will accept or inhibit the received signal, using the following formula:

$$\tau_k^j(\mathbf{x}) = p_j^k \bigwedge_{i=1}^n \bigwedge_{\ell \in L} (-1)^{1-\ell} (x_i + w_{i,j,k}^\ell), \quad (4.19)$$

where x_i denotes the input value of the neuron N_i , and $L = 0, 1$ denotes the set of terminal fibers that synapse on the dendrite k and p_j^k is the excitatory or inhibitory response of the k -th of M_j neurons. In general, weight vectors are given by $\mathbf{w}_{j,k}^1 = \mathbf{a}$ and $\mathbf{w}_{j,k}^0 = \mathbf{b}$ where $[\mathbf{a}, \mathbf{b}]$ is an interval.

The set of values computed at dendrites τ_k^j , $k = 1, \dots, K^j$, where K^j denotes the total number of dendrites of the neuron M_j , are processed at the cell body of M_j using one of

Figure 20 – Morphological Perceptron with Dendritic Structure



(a) Biological structure of a neuron.

(b) Graphical representation of an artificial neuron with dendritic computation.

Source: (RITTER; IANCU; URCID, 2003)

the following rules:

$$\tau^j(\mathbf{x}) = \bigwedge_{k=1}^{K^j} \tau_k^j(\mathbf{x}) \quad (4.20)$$

$$\tau^j(\mathbf{x}) = \bigvee_{k=1}^{K^j} \tau_k^j(\mathbf{x}) \quad (4.21)$$

Therefore, each neuron takes into account values processed on dendrites and it aggregates dendrites using an intersection or an union strategy using Equations 4.20 or 4.21, respectively. Then, the neuron M_j state is given by an activation function:

$$y_j = f(\tau^j(\mathbf{x})), \quad (4.22)$$

where f is the hard limited function given by Equation 4.2. Figure 20 illustrates a biological representation of neurons (a) and graphical neural pathways of the dendritic model (b).

Some Observations on MPs and DMMNs

Some remarks on the relation between MPs and DMMNs have been pointed out by Sussner and Campiotti (SUSSNER; CAMPIOTTI, 2020). It was stated that a simple way to link both theories is to understand each anti-dilation and erosion as an excitatory and inhibitory computation of neuron's dendrites, respectively. It was also shown that

Equations 4.14 and 4.19 are equivalent whenever the output response $p_j^k = 1$ is excitatory. Recalling that the computation of MP's unit is given by Equations 4.14 and 4.16. On the other hand, If we consider a DMNN unit of computation, DMNN that involves inhibitory output responses $p_j^k = -1$ are mappings between complete lattices and, therefore, can be expressed in terms of infimum of erosion and anti-dilation or in terms of supremum of anti-erosion and anti-dilation operations (BANON; BARRERA, 1993). In particular, assuming $p_j = 1$, we have:

$$\bigwedge_{i=1}^n \bigwedge_{\ell \in L} (-1)^{1-\ell} (x_i +' w_i^\ell) = \bigwedge_{i=1}^n (x_i +' w_i^1) \wedge \bigwedge_{i=1}^n (x_i^* +' (w_i^0)^*) \quad (4.23)$$

Hence, whenever $p_j^k = 1$ dendrite units of computation can be described in terms of erosion and anti-dilation operators $\varepsilon_{\mathbf{v}} \wedge \bar{\delta}_{\mathbf{w}}$ with the configuration $v_i = w_i^1$ and $w_i = (w_i^0)^*$, for $i = 1, \dots, n$.

If we consider units of computation of DMNNs with inhibitory output response $p_j^k = -1$, we have:

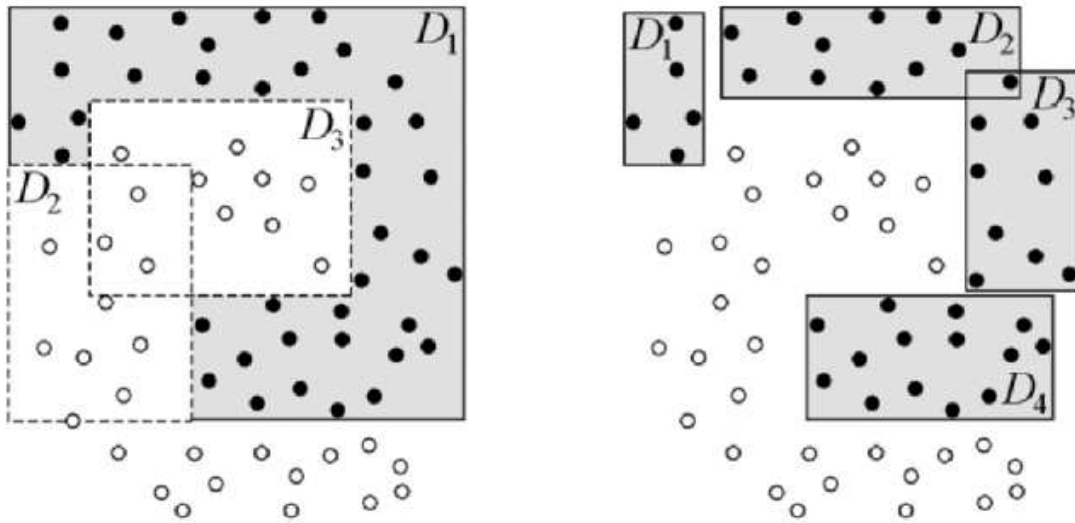
$$\begin{aligned} \left(\bigwedge_{i=1}^n (x_i +' w_i^1) \wedge \bigwedge_{i=1}^n (x_i^* +' (w_i^0)^*) \right)^* &= \left(\bigwedge_{i=1}^n (x_i +' w_i^1) \right)^* \vee \left(\bigwedge_{i=1}^n (x_i^* +' (w_i^0)^*) \right)^* \\ &= \left(\bigvee_{i=1}^n (x_i +' w_i^1)^* \right) \vee \left(\bigvee_{i=1}^n (x_i^* +' (w_i^0)^*)^* \right) \quad (4.24) \\ &= \left(\bigvee_{i=1}^n (x_i^* + (w_i^1)^*) \right) \vee \left(\bigvee_{i=1}^n (x_i + (w_i^0)) \right) \end{aligned}$$

Therefore, whenever $p_j^k = -1$, Equation 4.19 can be expressed in terms of dilation and anti-erosion operations: $\bar{\varepsilon}_{\mathbf{v}} \vee \delta_{\mathbf{w}}$ with the configuration $v_i = (w_i^1)^*$ and $w_i = (w_i^0)$, for $i = 1, \dots, n$. In this case, a neuron in a hidden layer compute an output according to rules described in Equations 4.11 and 4.12. Note that the conjugation operation extends multiplication by -1, which can be seen as an extension of the function $\mathbb{R} \rightarrow \mathbb{R}$, to a function $\mathbb{R}_{\pm\infty} \rightarrow \mathbb{R}_{\pm\infty}$ from the extended reals to the extended reals.

Consequently, feedforward DMNN models developed to supervised learning are MPs in the sense of Section 4.2 and 35. Moreover, Equation 4.24 shows that MP models have the biological interpretation of DMNNs.

Since MP computations are drawn from minimax algebra, whose operations are neither smooth nor differentiable. A training by conventional gradient descent or backpropagation is not possible. However, algorithms that incrementally build the parameters have been proposed. It becomes important to explore the properties and difference between these algorithms. Next, we present some notions and examples on constructive algorithms to train MPs.

Figure 21 – Elimination and Merge Principles



Class separation by elimination (left) and by merge (right).

Source: (RITTER; IANCU; URCID, 2003)

4.4 MP and DMNN Training Algorithms

A constructive MP training algorithm builds hyperboxes incrementally following some strategy. Next, a brief review on constructive algorithms for training MPs can be found. We summarize some constructive algorithms developed to find the hyperboxes and their properties. Besides, we review some algorithms that have proposed optimization methods to train MPs and DMNNs.

Elimination - Merge Algorithm

Two strategies to accomplish the class separation using the dendritic model were described in (RITTER; URCID, 2007). These strategies are also based in encapsulate pattern in hyperboxes and each one makes use of inhibitory and excitatory dendrites. Therefore, points that lie in a hyperbox $\underline{\mathbf{w}} \leq \mathbf{x} \leq \overline{\mathbf{w}}$ are computed as positive by excitatory dendrites ($p_j^k = 1$) with vector weights $[\underline{\mathbf{w}}, \overline{\mathbf{w}}]$, meanwhile inhibitory dendrites ($p_j^k = -1$) compute negative values for points that lies in the hyperbox.

Figure 21 illustrates a visual interpretation of these two perspectives for binary classification. The elimination algorithm (EA-MP) separates distinct classes by means of an intersection operation and, on the other hand, the merge algorithm uses a union operation. Hence, merging is performed by computing the union of the regions recognized by the dendrites and elimination is performed computing the intersection of regions.

On the left side of Figure 21, an elimination algorithm example is presented. Note that solid dots and circles represent observations of two different classes. In this case, dendrite D_1 forms a hyperbox and it has an excitatory response, thus it will process every enclosed

point as positive. Dendrites D_2 and D_3 process the values with an inhibitory response, therefore every point outside their respective hyperboxes will have positive values. The class C_1 representation is done by the intersection of these three dendrites.

The right side of Figure 21 illustrates an example of the merge algorithm (MA-MP). Here, the union of four dendrites D_1, D_2, D_3 and D_4 with an excitatory response make the class separation possible. We can note that neurons used in the merge algorithm, where every dendrite is excitatory, can be computed by the Equation 4.8 of multilayer generalized morphological perceptron. In this way, besides inhibitory dendrites, the contribution of dendrites is most descriptive. However, it is remarkable that two algorithms for multi-class classification based on merge and elimination were also described in (RITTER; URCID, 2007). In addition, the following result was also demonstrated.

Theorem 29. *If $\{X_1, \dots, X_m\}$ is a collection of disjoint compact subsets of \mathbb{R}^n and ϵ is a positive number with $\epsilon < \epsilon_0$, then there exists a single layer lattice perceptron that assigns each point $x \in \mathbb{R}^n$ to a class C_j whenever $x \in C_j$ and $j \in \{1, \dots, m\}$, and to class $C_i = \neg \bigcup_{j=1}^n C_j$ whenever $d(x, X_i) > \epsilon$, $\forall i = 1, \dots, m$. Furthermore, no point x is assigned to more than one class.*

Therefore, the convergence and a perfect class-separation is guaranteed.

MP/CL Algorithm

A learning algorithm that uses only dendrites with excitatory output responses is discussed in (SUSSNER, 1998). It produces areas of indecision and may assign multiple labels to the same output. In addition, the decision surface depends on the order in which the training patterns are presented to the network. A new algorithm, called MP/CL that overcome these disadvantages was presented for binary classification (SUSSNER; ESMI, 2009b; SUSSNER; ESMI, 2011). The pseudo-code of MP/CL for binary classification is presented in Algorithm 1.

Consider the set of training data $\mathcal{T} = \{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^k, y^k)\}$ where $x^i \in \mathbb{R}^n$ and y^i is its desired class-label. The set $X = \{\mathbf{x}^1, \dots, \mathbf{x}^k\}$ denotes the observation. Lets \mathcal{T}^s denote the patterns associated with the class-label s and $s = 1, 2$. The symbol \mathcal{C} denotes the pattern that are considered misclassified which is initialized as $\mathcal{C} = X$. The set of hyperboxes of the s class is denoted \mathcal{F}_s , for $s = 1, 2$. Finally, two half-spaces denoted by $H_i^-(c)$ and $H_i^+(c)$ and the hyperplane $P_i(c)$ are given by:

$$H_i^-(c) = \{\mathbf{x} \in \mathbb{R}^n : x_i < c_i\}, \quad H_i^+(c) = \{\mathbf{x} \in \mathbb{R}^n : x_i > c_i\} \quad \text{and} \quad P_i(c) = \{\mathbf{x} \in \mathbb{R}^n : x_i = c_i\}$$

In addition, Algorithm 1 was extended to multiclass following a one vs all classes strategy. Moreover, the next properties were demonstrated for MP/CL algorithm:

Algorithm 1 – Training Algorithm MP/CL

```

begin
   $\mathcal{C} = X$  and  $\mathcal{F}_s = \emptyset$ , for  $s = 1, 2$ ;
  while  $\mathcal{C} \neq \emptyset$  do
    (Select an arbitrary subset  $\bar{X}$ )
     $\bar{X} \subseteq \mathcal{C}$  and  $\bar{C}^s = \bar{X} \cap C^s$  for  $s = 1, 2$ ;
    For  $s = 1, 2$ ,  $[\mathbf{a}^s, \mathbf{b}^s] = [\bigwedge \bar{C}^s, \bigvee \bar{C}^s]$ ;
     $I = [\mathbf{a}, \mathbf{b}] = [\mathbf{a}^1, \mathbf{b}^1] \cap [\mathbf{a}^2, \mathbf{b}^2]$ ;
    if  $I = \emptyset$  then
       $\mathcal{C} = \mathcal{C} \setminus \bar{X}$ ;
       $\mathcal{F}_s = \mathcal{F}_s \cup [\mathbf{a}^s, \mathbf{b}^s]$  for  $s = 1, 2$ ;
    end
    else
      if  $[\mathbf{a}^1, \mathbf{b}^1] \neq [\mathbf{a}^2, \mathbf{b}^2]$  then
         $\mathcal{F}_s = \mathcal{F}_s \cup [\bigwedge (H_i^-(\mathbf{a}) \cap \bar{C}^s, \bigvee (H_i^-(\mathbf{a}) \cap \bar{C}^s)], s = 1, 2$  and
         $\mathcal{F}_s = \mathcal{F}_s \cup [\bigwedge (H_i^+(\mathbf{b}) \cap \bar{C}^s, \bigvee (H_i^+(\mathbf{b}) \cap \bar{C}^s)], s = 1, 2$ ;
        (After updating the families of hyperboxes  $\mathcal{F}_s$ , we substitute  $\bar{X}$  with
         $\bar{X} \cap I$  in  $\mathcal{C}$ , if  $\bar{X} \cap I = \emptyset$ ;

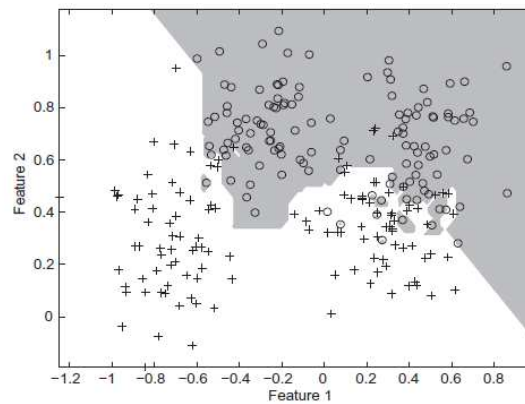
         $\mathcal{C} = \mathcal{C} \setminus \bar{X}$ 
      end
      if  $I = [\mathbf{a}^1, \mathbf{a}^1] = [\mathbf{a}^2, \mathbf{a}^2]$  then
         $x = \mathbf{b}$  (An arbitrary vertex);
         $\mathcal{P} = \{P_i(x) : |P_i(b) \cap \bar{X}| \leq |P_j(b) \cap \bar{X}| \text{ for } i, j = 1, \dots, n\}$  and
         $U = \bigcup_{P, P' \in \mathcal{P}, P \neq P'} (P \cap P')$ ;
         $\mathcal{F}_s = \mathcal{F}_s \cup \{\{\mathbf{u}\} : \mathbf{u} \in U \cap C^s\}, s = 1, 2$  and  $\bar{X} = \bar{X} \setminus \bigcup \mathcal{P}$ ;
         $\bar{\mathcal{P}} = \{P \setminus U : P \in \mathcal{P}\}$ ;
         $\mathcal{C} = \mathcal{C} \cup \{\bar{X} \cap \bar{P} : \bar{P} \in \bar{\mathcal{P}} \text{ and } \bar{X} \cap \bar{P} \neq \emptyset\}$ 
      end
    end
  end
end

```

1. Convergence in a finite number of steps.
2. Perfect separation of the training data according to their class labels.
3. Hyperboxes with distinct class labels do not overlap.
4. Independence of the order in which the training patterns are presented to the network.

The MP/CL exhibited a good performance and fast convergence without requiring any fine tuning of weights. Beside a MP/CL is able to produce complex decision surfaces

Figure 22 – MP/CL’s Decision Surface Example



Decision surface produced by the MP/CL training algorithm for Ripley’s synthetic problem. Source: (SUSSNER; ESMI, 2011)

to classify efficiently non-linearly separable data. Figure 22 illustrates an example of a decision surface produced by the MP/CL.

Divide and Conquer

The divide and conquer algorithm (D&C) was first presented by Arce et al., as an efficient training algorithm to train DMNN for supervised classification (ARCE et al., 2017). The idea of this algorithm is to recursively divide hyperboxes into smaller regions (which are also hyperboxes) and check how the a hyperbox is performed. When all these smaller hyperboxes classify only a class j correctly, the whole region is considered a new hyperbox of j . Applications and simulations that employ this algorithm can be found in (ARCE et al., 2017). A drawback of this algorithm is that it computes too many regions. Besides, when the dataset is sparse, a large number of hyperboxes can be computed and overfitting can occur. Furthermore, this process may be expensive in terms of memory and computation. Therefore, a memory problem could happen whenever dimension n is high. Pseudo-code that shows the process of this method can be found in Algorithm 2. The reader may notice that D&C algorithm makes use of post-synaptic excitatory dendrites.

Differential Evolution Training

A training algorithm that makes use of an evolutionary approach was introduced in (ARCE et al., 2018), we will refer to this model as DE-MP. In this work, only excitatory post-synaptic dendrites are used. The idea consists in initializing a population of hyperboxes using the K++ means algorithm (ARTHUR; VASSILVITSKII, 2006). It is for each class a fixed q number of hyperboxes is computed, automatically increasing the number of

Algorithm 2 – Divide and Conquer Training Algorithm

```

begin
  Initialize a box  $B$  that encloses all the patterns;
  Add  $B$  to a list of boxes  $Bs$ ;
  while  $Bs \neq \emptyset$  do
     $S = Bs(1)$ ;
    if  $S$  only contains elements of a class  $j$  then
      | Add the hyperbox  $S$  to the list of boxes of the class  $\mathcal{F}^j$ ;
    end
    else
      | Divide  $S$  into  $2^n$  smaller hyperboxes and add them to  $Bs$ ;
    end
    Remove  $S$  from  $Bs$ .
  end
end

```

hyperboxes when the hyperbox error is too high.

The differential algorithm used to train MPs has the following components.

1. **Initialization** method to generate the initial population. Each individual is a set of hyperboxes.
2. The **fitness function** to minimize the error. Particularly, it is used the number of misclassified patterns divided by the total number of patterns(%).
3. The **mutation** operation performs a permutation of 30% of the rows, in order to change the order of the classes.
4. A **Differential evolutionary function** creates a new populations using:

$$L_t = L_a + F(L_b - L_c) \quad (4.25)$$

where L_a are the best individual at some epoch, L_b and L_c are lists of random elements of the same epoch.

5. The **stopping conditions** are a fixed number of epochs or an error lesser that some ϵ .

The pseudo-code of this method is given in Algorithm 3.

Although DE algorithm has shown good experimental results in some simulations, it does not guarantee perfect separation of classes or the same result independently of the order in which the training patterns are presented to the network. However, a training algorithm that makes use of iterative non-linear optimization techniques such as differential evolutionary methods can get automatized solutions that could outperform other constructive algorithms

Algorithm 3 – Differential Evolution Training Algorithm

```

begin
  while Stop Conditions do
    for  $d = 1, \dots, q$  do
      Compute the Best Initial Population (K++ Means);
      Compute the fitness value;
      Select  $s$  solutions at random and the best parents  $s/2$  parents;
      Create  $s/2$  offspring using DE equation (Equation 4.25);
      for each offspring d1 do
        If the fitness value is better for the offspring, worst parent is
        replaced by d1.
      end
    end
  end
end

```

in the test stage, it is, outperform in terms of solution generalization. Therefore, we can observe that we considered this algorithm as constructive, although it uses an optimization method to minimize a loss function. This follows the fact that it builds, incrementally, hyperboxes based on the k++ means algorithm.

Tuning the Weights of Morphological Perceptrons using SGD Optimization

A method that involves SGD to train morphological perceptrons in terms of DMNN was developed by Zamora and Sossa (ZAMORA; SOSSA, 2017). However, this method was not only based on SGD, since it uses constructive methods to initialize hyperboxes. We will refer to this model as DMNN-SGD.

In order to update the hyperboxes parameters and minimize a loss function, some modifications were proposed to the original unit of computation of MPs (see Equation 4.19).

$$\tau_k^j(\mathbf{x}) = \bigwedge_{i=1}^n (w_{ki}^j + b_{ki}^j - x_i) \wedge (x_i - w_{ki}^j) \quad (4.26)$$

In this case, instead of enclosing a pattern using interval-valued parameters $[\underline{\mathbf{w}}, \overline{\mathbf{w}}]$, a parameter $\mathbf{b} \in \mathbb{R}_+^n$ that represents the length of an interval is used. Therefore, an implicit interval-valued representation $[\mathbf{w}, \mathbf{w} + \mathbf{b}]$ is used to perform the basic computation of a neuron. In addition, the competition layer is replaced by a softmax layer. Thus, a cross entropy loss function is attached to this model. We can consider that this model uses only excitatory output responses.

The probability that a pattern \mathbf{x} belongs to the class j is given by $P_j(\mathbf{x})$. The cross entropy between these probabilities and the targets (ideal probabilities) can be used as the target

function J for a mini-batch $Z = \{\mathbf{x}_1, \dots, \mathbf{x}^Q\}$:

$$J(Z) = \sum_{q=1}^Q \sum_{j=1}^{n_c} \{t_q = j\} \log(P_j(\mathbf{x}^q)) \quad (4.27)$$

the gradient search direction for the parameters w_{ki}^j and b_{ki}^j are:

$$\frac{dJ}{dw_k^j} = - \sum_{q=1}^Q [0 \ 0 \ \dots \ f_{kj} \ 0 \ 0] \quad (4.28)$$

and

$$\frac{dJ}{db_k^j} = - \sum_{q=1}^Q [0 \ 0 \ \dots \ f_{kj} \ 0 \ 0] \quad (4.29)$$

Note that the gradient directions are computed using only those parameters that contributes to the output response, the others are ignored. Besides, search directions are set to zero in the places where partial derivatives do not exist. Thus, we can observe that, in the best case, each pattern \mathbf{x} contributes with a single hyperbox parameter w_{ki}^j modification for each class. Therefore, a random initialization of the weights of this model is not possible. To overcome this drawback, it was suggested to initialize the weights using a constructive method. Consequently, this method is used to improve weights that were previously computed by other method. Also, the algorithm can lead to overtraining whenever a constructive algorithm that already guarantee a perfect training is considered to initialize the weights. Note that this algorithm does not determine automatically the final number of hyperboxes or perfect recall. Experimentally, some simulations with promising results were presented in (ZAMORA; SOSSA, 2017).

Ellipsoidal Dendritical Neural Networks

A morphological perceptron that replaces hyperboxes with hyper-ellipsoids was introduced by Arce et al., (ARCE et al., 2019). We will refer to this model as ellipsoidal DNN (Ell-DNN). Furthermore, a training procedure based on k++ means was presented. In this model the basic computation of a dendrite is given by:

$$\tau_k^j(\mathbf{x}) = (\mathbf{x} - \mu_k)^T A_k^{-1} (\mathbf{x} - \mu_k) \quad (4.30)$$

where μ is the mean vector related with the k -th hyper-ellipsoid of the j -th neuron and A_k^{-1} is a covariance matrix. The value of the j -th neuron is given computing the infimum of K hyper-ellipsoids as follows:

$$\tau^j(\mathbf{x}) = \bigwedge_{k=1}^K \tau_k^j(\mathbf{x}) \quad (4.31)$$

where K is also the number of dendrites of the neuron j -th. In this case, if $\tau^j(\mathbf{x}) = 0$ then \mathbf{x} is in the center of some hyper-ellipsoid $k = 1 \dots K$, if $\tau^j(\mathbf{x}) > 1$ then \mathbf{x} is outside of all

the hyper-ellipsoids $k = 1 \dots K$. This is, as the inputs move away from the hyper-ellipsoid centroid, the output value grows. The process to train this model is given in Algorithm 4. Note that it is needed to compute matrix inverses in this procedure. This can be taken as

Algorithm 4 – Ellipsoid Training Algorithm

```

begin
  Given an initial number of dendrites per class  $K$ . Initialize a global error  $e_g$ ;
  for each class  $C_j$  do
    Initialize  $e_k = \infty$  set  $K_l = K$  while  $e_k \geq e_g$  do
      Generate  $K_l$  clusters using K++-means;
      for each cluster  $k = 1, \dots, K$  do
        Compute  $\mu_k$  and  $\sum_k$ ;
        Compute the inverse  $\sum_k^{-1}$ ;
        Calculate the a hyperbox error for the class  $e_k$ ;
        if  $e_k \geq e_g$  then
          |  $K_l = K_l + 1$ 
        end
      end
    end
  end
end
end

```

a drawback in two senses. Since matrix multiplications and inverses are involved in each dendrite computation, the complexity of this algorithm is high. Besides, to store all the matrices could be a problem in terms of memory. This method has been tested in few simulations (ARCE et al., 2018). Nevertheless, in these particular problems it has shown a competitive performance.

4.5 Hybrid Morphological Models

Morphological/Rank/Linear Networks

Morphological/Rank/Linear Neural Networks (MRL-NN) were first described by Pessoa and Maragos (PESSOA; MARAGOS, 2000) as a multilayer feedforward network that combines linear filters and operators of morphological perceptrons. Particularly, the MRL-NN model combines morphological/rank neural networks (PESSOA; MARAGOS, 1996) (MRNN) and classical MLP operations. The unit of computation of an MRL-NN is

called MRL-NN filter and it computes the convex combination:

$$\begin{aligned} y &= \lambda\alpha + (1 - \lambda)\beta, \\ \alpha &= R_r(\mathbf{x} + \mathbf{a}) = R_r(x_1 + a_1, \dots, x_n + a_n), \\ \beta &= \mathbf{x} \cdot \mathbf{b}^T + t, \end{aligned} \quad (4.32)$$

where $\lambda \in [0, 1]$, $\mathbf{x}, \mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ and the function $R_r(\mathbf{z})$ is evaluated sorting the components z_i in decreasing order and picking the r -th element of the sorted list. The particular cases R_1 and R_n become a morphological dilation and erosion, respectively. Therefore, the vector \mathbf{a} can be considered a structuring element. These units of computations are organized as a layer. Note that whenever $\lambda = 0$ the unit of computation becomes a classical computation of an MLP node, and if $\lambda = 1$ it becomes a computation of a MRNN. Therefore, this model can be viewed as an extension of the classical perceptron model.

In their work, besides a solution to the parity problem using an MRL-NN model, Pessoa and Maragos published an algorithm that modifies conventional BP to train MRL-NN nodes. Observe that among all the functions involved in the computation of Equation 4.32, the rank filter is not differentiable. To deal with this problem, it was proposed to use pulse functions (PESSOA; MARAGOS, 2000) and to compute parameters r in terms of the function:

$$r = \left\lceil n - \frac{N_{l-1} - 1}{1 + \exp(-p)} + 0.5 \right\rceil, \quad (4.33)$$

where n is the dimension of the output of the previous layer. This is an elegant way to map from a variable p to an integer r . In this case, if $p \rightarrow -\infty$ then R_r corresponds to a minimum operation, if $p = 0$ then it is equivalent to a median and if $p \rightarrow \infty$ then it represents the maximum operation. The set of weights for a node are represented by the following vector $\mathbf{w} = (\mathbf{a}, p, \mathbf{b}, t, \lambda)$. The partial derivatives with respect to p , which is the one parameter involved with the rank filter, are given by:

$$\frac{\partial \alpha}{\partial p} = 1 - \frac{1}{1 - N_{l-1}} Q(\alpha \mathbf{1} - \mathbf{x} - \mathbf{a}) \cdot \mathbf{1}, \quad (4.34)$$

where \mathbf{x} are the inputs coming from the previous layer and $Q(\mathbf{v}) = (q(v_1), \dots, q(v_n))$ where $q(v_i) = 1$, if $v_i = 0$ and $q(v_i) = 0$, otherwise. The linear and convex parts of the gradients can be derived as usual, using the chain rule. A common issue addressed in of this approach is exploding gradients. Therefore, abrupt changes in the gradient estimation can lead to numerical stability problems. Nevertheless, a simple solution to this problem can be implemented using a smoothed version of $q(v_i)$ such as $q_\sigma(v_i) = \exp(-\frac{1}{2}v_i/\sigma)^2$, where $\sigma > 0$ is a smoothing factor.

Dilation Erosion Linear Perceptron

The Dilation Erosion Linear Perceptron (DELP) was initially proposed by Araújo et al., in (ARAÚJO; OLIVEIRA; MEIRA, 2012), a model composed of a combination of morphological and linear operators was presented. The learning process of DELP

involves a backpropagation based method where the non-differentiability problem of the morphological part is systematically circumcised using some observations from MRNN framework and extending the ideas introduced in (ARAÚJO, 2011).

Each DELP unit of computation is represented by a balanced combination between dilation and erosion operators and an activation function. The unit of computation outputs of a DELP layer are given by:

$$\begin{aligned}
 y_n &= f(u_n) \text{ for } n = 1, \dots, N_l, \\
 u_n &= \lambda_n \delta_n(\mathbf{x}) + (1 - \lambda) \varepsilon_n(\mathbf{x}), \quad \lambda \in [0, 1], \\
 \delta_n(\mathbf{x}) &= \bigvee_{i=1}^n (x_i + a_{n,i}), \\
 \varepsilon_n(\mathbf{x}) &= \bigwedge_{i=1}^n (x_i + b_{n,i}),
 \end{aligned} \tag{4.35}$$

where $\mathbf{x} \in \mathbb{R}^n$ are the inputs coming from the previous layer and $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$ are the dilation, erosion weights, respectively. A common choice for activation function f is the sigmoid function (see Table 2).

Following the strategy of (PESSOA; MARAGOS, 1996), a vector of weights $\mathbf{w}_n = (\lambda, \mathbf{a}_n, \mathbf{b}_n)$ must be updated using backpropagation for each unit of computation in a DELP layer. The main issue is the lack of differentiability of the dilation and erosion operations. However, as it was mentioned, these operations can be regarded as particular cases of rank operators.

$$\bigvee_{i=1}^n (x_i + a_{n,i}) = R_1(\mathbf{x} + \mathbf{a}) \quad \bigwedge_{i=1}^n (x_i + b_{n,i}) = R_n(\mathbf{x} + \mathbf{a}) \tag{4.36}$$

To deal with abrupt changes in gradient estimation, DELP also make use of smoothed pulse functions with the form $q_\sigma(v_i) = \frac{1}{2} \exp(\frac{v_i}{\sigma})$. Following these considerations, gradients of parameters can be computed as follow:

$$\begin{aligned}
 \frac{\partial \varepsilon_n}{\partial \mathbf{a}_n} &= \frac{Q_\sigma(\varepsilon_n \cdot \mathbf{1} - [x + b_n])}{Q_\sigma(\varepsilon_n \cdot \mathbf{1} - [x + b_n]) \cdot \mathbf{1}}, \\
 \frac{\partial \delta_n}{\partial \mathbf{a}_n} &= \frac{Q_\sigma(\delta_n \cdot \mathbf{1} - [x + a_n])}{Q_\sigma(\delta_n \cdot \mathbf{1} - [x + a_n]) \cdot \mathbf{1}}, \\
 \frac{\partial u_n}{\partial \lambda_n} &= \delta_n(\mathbf{x}) - \varepsilon_n(\mathbf{x})
 \end{aligned} \tag{4.37}$$

These partial derivatives are used to compute the gradients. Then, a standard BP algorithm can be applied.

The Convex-Concave Procedure

The Convex-Concave procedure (CCP) optimizes functions subject to weak conditions that can be expressed as the sum of a concave and a convex part. CCP has been

used to train binary classifiers such as support vector machines. This algorithm makes use of gradients to optimize parameters and perform the minimization together with a modified version of SGD. The idea is to build a convex cost function with constraint rules that consist of difference-of-convex functions (DC). Then, the CCP is a general principle which can be used to construct discrete time iterative dynamical systems for almost any energy minimization problem (YUILLE; RANGARAJAN, 2002).

Charisopoulos et al., presented how to use this algorithm to train a classifier, which is in fact, a modified morphological perceptron that makes use of erosion and dilation operators (CHARISOPOULOS; MARAGOS, 2017).

Lets C_0, C_1 denote the class-labels of a binary hyperbox and let $x \in \mathbb{R}^n$ be a pattern.

$$\begin{aligned} \text{Minimize } J(X, w) &= \sum_{j=1}^K \max(\xi_j, 0) \\ \text{s.t. } &\begin{cases} \bigvee_{i=1}^n w_i + x_i^k \geq \xi_j & \text{if } x^k \in C_0 \\ -\bigvee_{i=1}^n w_i + x_i^k \geq \xi_j & \text{if } x^k \in C_1 \end{cases} \end{aligned} \quad (4.38)$$

The solution to this construction can be approximated by the CCP. The set of slack variables ξ_k in the constraints are used to ensure that only misclassified observations contribute to the cost function. However, in this approach (and most MPs based approaches), outliers and feature extraction are not considered. To overcome this problem, (CHARISOPOULOS; MARAGOS, 2017) also proposed to penalize those patterns with greatest chances of being outliers. The penalization is computed using the distance (ℓ_p) between patterns and class centroids.

$$\mu_i = \frac{1}{|C_i|} \sum_{x^k \in C_i} x^k \quad (4.39)$$

$$\lambda_k = \frac{1}{\|x^k - \mu_i\|_p} \quad (4.40)$$

$$v_k = \frac{\lambda_k}{\max_k \lambda_k} \quad (4.41)$$

So the minimization can be performed using:

$$\begin{aligned} \text{Minimize } J(X, w) &= \sum_{j=1}^K v_k \cdot \max(\xi_j, 0) \\ \text{s.t. } &\begin{cases} \bigvee_{i=1}^n w_i + x_i^k \geq \xi_j & \text{if } x^k \in C_0 \\ -\bigvee_{i=1}^n w_i + x_i^k \geq \xi_j & \text{if } x^k \in C_1 \end{cases} \end{aligned} \quad (4.42)$$

This method is referred as WD_{CCP} . Finally, the response is given by a dilation-erosion linear combination.

$$y(\mathbf{x}) = \lambda \left(\bigvee_{i=1}^n w_i + x_i \right) + (1 - \lambda) \left(\bigvee_{i=1}^n w_i + x_i \right) \quad (4.43)$$

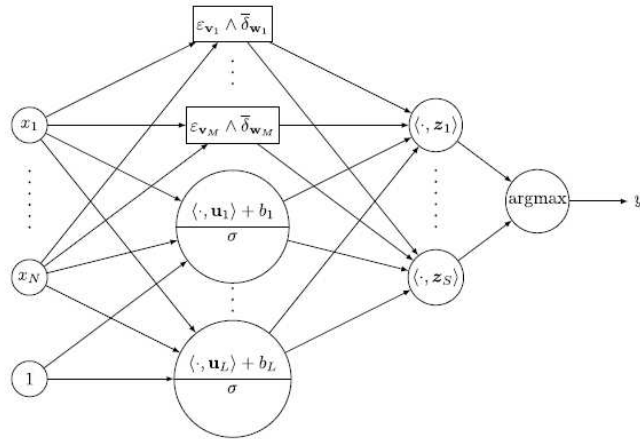
Experiments and simulations using gradient descent with an MSE (Equation 3.3) and WD_{CCP} were conducted and described in (CHARISOPOULOS; MARAGOS, 2017).

Hybrid Morphological Linear Perceptron

A novel Hybrid Morphological Linear Perceptron (HMLP) model was presented in (SUSSNER; CAMPIOTTI, 2020). In this approach, a set of linear and morphological nodes are combined in the same layer. This proposal differs from the previous models in several aspects. For example, instead of computing convex combinations of linear and morphological operators at each node, individual nodes compute a normalized linear or morphological operation of the form given by Equation 4.14. Also, HMLP does not compute pulse function operations to calculate gradients of the morphological part. That is, instead of training the morphological nodes at the hidden layer, it was proposed to make use of extreme learning machine (HUANG; ZHU; SIEW, 2006; HUANG, 2014; HUANG, 2015). The HMLP architecture is illustrated in Figure 23. The HMLP model is a feedforward two-layer model with a hidden layer with morphological units as well as classical semi-linear neurons with sigmoid activation functions (See Table 2). An output layer with only semi-linear neurons performs the separation using the data of the hidden layer.

The HMLP model combines the approximation capabilities of the two-layer perceptron

Figure 23 – HMLP Topology



Source (SUSSNER; CAMPIOTTI, 2020)

having sigmoid activation functions with the capability of the MP to represent non-differentiable functions. Hence, HMLP is a single hidden layer computational model in which, morphological nodes compute $\varepsilon_{\mathbf{v}}(\mathbf{x}) \wedge \bar{\delta}_{\mathbf{w}}(\mathbf{x})$ and linear nodes work as usual with a slight modification, a constant term that regularizes the output of linear nodes.

The proposed process to train an HMLP uses extreme learning machine and therefore it initializes randomly the hidden layer nodes (both morphological and semi-linear). Then,

weights at the output layer are trained using a gradient descent approach (HUANG, 2015). The resulting model is called HMLP-EL and it has shown very interesting properties in a hyperbox experiments (SUSSNER; CAMPIOTTI, 2020).

4.6 Challenges and Issues of Morphological Perceptrons

To analyze the algorithms studied in the last section, we can start checking the general properties that they satisfy. First, we can observe that all the models and algorithms studied in the last section enclose patterns of a certain class in some geometrical structures. For example, Ell-DNN presented in 4.4 makes uses of hyper-ellipsoids. Another interesting property, which is related with the dendrite biological interpretation given in 4.3, is the use of post-synaptic excitation and inhibition response at dendrites. As example, elimination algorithm makes use of both types of post-synaptic responses, which means that it uses hyperboxes that enclose patterns that do not belong to a class. These algorithms can also be constructive, trained by some optimization method or use an hybrid approach in which a constructive and an optimization method are used together. Example of hybrid approaches are the DE-MP and the DMNN-SGD.

To guarantee a perfect recall on the training data and to get the same boundary decision independently to the order in which the training data is presented, are desirable properties for MP algorithms. Table 4 summarizes an analysis that considers those properties. In

Table 4 – Algorithms Properties

	EO	IO	C	O	PR	LB	DO-I	Geometry
MA-MP	X		X		X	X		Hyperboxes
EA-MP	X	X	X		X	X		Hyperboxes
MP/CL	X		X		X	X	X	Hyperboxes
D&C	X		X		X	X		Hyperboxes
WD- <i>CCP</i>	X		X	X	X			Polytopes
DMNN-SGD	X		X	X		X		Hyperboxes
Ell-DNN			X		X			Hyper-Ellipsoids

EO and IO denote dendrites with excitatory and inhibitory response, respectively. C and O stand for methods with constructive and optimization based algorithms. PR denotes perfect recall for training dataset and DO-I means that the output is independent of the data order. Lattice based method is denoted by LB and geometry denotes the basic geometric-shape structure used by models.

general, MP training algorithms are highly geometrical procedures that take an input and build the separation without any natural feature pre-processing. Unlike, neural networks in which features and characteristics are automatically discovered from raw inputs and then classified, MPs do a hyperbox directly from the raw inputs.

With the exception of the WD_{CCP} model, which uses a penalization to discard negative effect from outliers, MP training algorithms do not have attached any special system to analyze relation between class attributes, outliers, or pattern frequencies. This disadvantage can be overcome including other operations and operators into the MP framework. Those operation should have some special properties that allow a natural link between the geometrical idea of MP and other techniques of processing.

In the following chapter, we will extend the framework of MPs, hopefully, it will allow us to attach a natural feature-extractor or pre-processing techniques that improve the quality of the hyperboxes computed by some algorithm. Besides, new MP models will be detailed.

5 Generalized Morphological Perceptrons

Numbers never lie, after all:
they simply tell different stories
depending on the math of the
tellers.

Luís Alberto Urrea, The Devil's
Highway

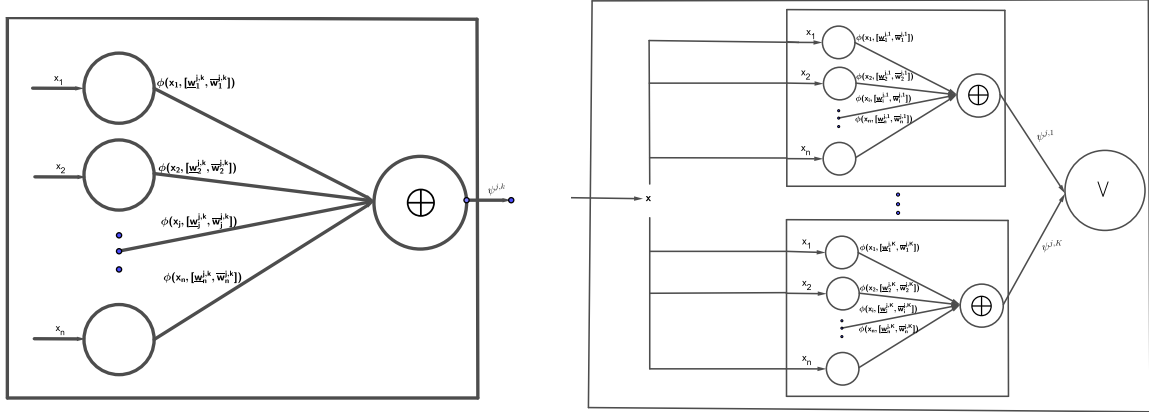
Indisputably, classifiers come with their geometries that determine their classification capabilities and limitations. Therefore, the relative position of patterns belonging to different classes in the feature space is a central issue in classification. Linear classifiers, whose geometry relies on hyperplanes basis, non-linear classifiers that make possible to separate instances that are not linearly separable or support vector machines that rely on transform inputs to higher dimensions and classify the instances according to these transformations are clear examples of different perspectives to accomplish the same task. Interval calculus has been widely used to develop classifiers that dwell on the concept of information granules. In classification based on granular information, it is assumed that patterns that belong to the same class form information granules in the feature space. Most classifiers that make use of granular information rely on the usage of techniques of granular computing. In particular, intervals can be used to build hyperbox-driven classifiers (PEDRYCZ; PARK; OH, 2008).

Many MP models and algorithms presented in Section 4.4 are hyperbox-driven, in which intervals play an important role. The learning rules attached to its training rely on how the elements of a certain class are enclosed by some geometric structure. Hence, it is in the best of our interest to study the behavior of intervals in the framework of ℓ -groups and ℓ -rings using \mathbb{I} -set descriptors.

Next, we introduce another perspective to analyze and describe some models presented in the last chapter. In order to achieve this task, we make use of lattice ordered structures. Instead of describing the operations in terms of $\mathbb{R}_{\pm\infty}$ and mathematical morphology, we will focus our attention on operations that naturally arise from ordered structures studied in Section 2.3. In particular, the use of Archimedean σ -rings allows to describe new units of computation that can be analyzed.

Besides, the biological interpretation of DMNN given in Section 4.3 is reviewed and adapted to this framework.

Figure 24 – Generalized Morphological Perceptron



- (a) Process of the k -th sub-module of the j -th class.
- (b) Process performed by the module of the j -th class with K sub-modules, using the union strategy and an identity activation function.

5.1 General Description of a GMP Model

In Generalized Morphological Perceptron (GMP), the compatible order and operators of ℓ -groups and ℓ -rings are the main part of GMP individual values.

Lets \mathbb{A}^n be an Archimedean o-ring, the k -th dendrite output of the j -th neuron is given by:

$$\psi_k^j(\mathbf{x}) = p_j \bigoplus_{i=1}^n \phi(x_i, [\underline{w}_i^{j,k}, \overline{w}_i^{j,k}]) \quad (5.1)$$

where $p_j \in \{-1, 1\}$ for inhibitory/excitatory output response, $\mathbf{x} \in \mathbb{A}^n$, $[\underline{w}_i^{j,k}, \overline{w}_i^{j,k}] \in \mathbb{I}_{\mathbb{A}}$, ϕ is an $\mathbb{I}_{\mathbb{A}}$ -set descriptor and \bigoplus is an aggregation function.

Therefore, a single dendrite $k = 1, \dots, K$ of a neuron measures some kind of relation between a pattern \mathbf{x} and an interval $[\underline{w}^{j,k}, \overline{w}^{j,k}]$. The j -th neuron can aggregate the outputs of these dendrites using a union or an intersection strategy, following the next rule:

$$\psi^j(\mathbf{x}) = \bigwedge_{k=1}^K \psi_k^j(\mathbf{x}) \quad (5.2)$$

$$\psi^j(\mathbf{x}) = \bigvee_{k=1}^K \psi_k^j(\mathbf{x}) \quad (5.3)$$

Finally, an activation function determines the final output of the j -th neuron, as follows:

$$y_j = f(\psi^j(\mathbf{x})) \quad (5.4)$$

We let the activation function f be model dependent, instead of attaching some standard activation function. 24a shows the data flow of a single sub-module j -th of k -th class module. The final output is given by a winner-takes-all competition. Given a set of labels L , class modules compete among each other to determine the assigned class label. Thus,

the assigned label to an input observation is given by a maxnet as follows:

$$o(\mathbf{z}) = \operatorname{argmax}_{k=1}^{|\mathcal{L}|} y^k(\mathbf{z}) \quad (5.5)$$

Particularly, basic computations of the model described in Section 4.3, which are given by Equation 4.19 can be expressed in terms of the inf-aggregation and inf-descriptors, as follows:

$$\begin{aligned} \tau_k^j(\mathbf{x}) &= p_j^k \bigwedge_{i=1}^n \bigwedge_{\ell \in \{0,1\}} (-1)^{1-\ell} (x_i - w_{i,j,k}^\ell) = p_j^k \bigwedge_{i=1}^n (w_{i,j,k}^0 - x_i) \wedge (x_i - w_{i,j,k}^1) \\ &= p_j^k \bigwedge_{i=1}^n \phi_\wedge(x_i, [w_{i,j,k}^1, w_{i,j,k}^0]) \end{aligned} \quad (5.6)$$

which implies that algorithms expressed in terms of DMNN can be also expressed in terms of \mathbb{I} -set descriptors and aggregation functions.

Also, the MP models presented in 4.2 can be described in terms of aggregations and \mathbb{I} -set descriptors. In particular, their basic computation, given by Equation 4.14 can be expressed as follows:

$$\begin{aligned} g(\mathbf{x}) &= \varepsilon_{\mathbf{a}^*}(\mathbf{x}) \wedge (\bar{\delta})_{\mathbf{b}}(\mathbf{x}) = \left[\bigwedge_{i=1}^n (x_i - a_i) \right] \wedge \left[\bigwedge_{i=1}^n (b_i - x_i) \right] \\ &= \bigwedge_{i=1}^n (x_i - a_i) \wedge (b_i - x_i) = \bigwedge_{i=1}^n \phi_\wedge(x_i, [a_i, b_i]), \end{aligned} \quad (5.7)$$

where $a_i \leq b_i$ for $i = 1, \dots, n$. We conclude that the MP model described by means of erosion and anti-dilation can be also expressed in terms of the inf-aggregation and inf-descriptor with $p_j^k = 1$. Therefore, any algorithm expressed in terms of the model reviewed in 4.2 can be described using operations of GMP.

Now that we have reviewed and unified these models into GMP, a first question that naturally arise is: what will bring us this new perspective?

5.2 A Review on Aggregator and Descriptor Combinations

A first observation is that pairs consisting of an inf-aggregation and an inf-descriptor produce non-differentiable and non-smooth functions. To overcome this drawback, we can define new models that at least are differentiable using other aggregations and \mathbb{I} -set descriptors.

In this chapter, we will review some properties on aggregations-descriptors pairs that perform a unit computation of a GMP model (Equation 5.1).

The first task is to find which pairs of aggregations-descriptors from those studied in Section 2.6 can be used with SGD. Let us start discarding those pairs that does not bring something new. For example, we studied in Section 2.6 that pairs involving inf-aggregations

are not smooth and also, not differentiable everywhere. Also, following the equality given in Equation 5.7, the pair inf-aggregation/inf-descriptor has been widely studied in DMNN and MPs. Another result about the inf-descriptor and inf-aggregation pair is given in Section 2.6. Another result about the inf-descriptor and inf-aggregation pair is given in Section 2.6. In particular, Theorem 24 links the usual d_∞ distance with the computation of MPs.

As stated in Theorem 22, if we consider the pair inf-aggregation, dot-descriptor, it does not bring to us a different geometry from the one generated by the inf-aggregation and inf-descriptor pair. Therefore, we can discard also this pair, which is neither smooth nor differentiable.

On the other hand, sum-aggregations had not been the focus of MP models. Besides, the dot-descriptor has not even been considered in terms of MPs. Hence, to use pairs sum-aggregation, dot-descriptor to build GMP units of computation is a good choice.

5.3 Generalized Morphological Perceptron in Deep Learning

Considering that basic structures of deep learning, studied in Section 3.3, are layers, it is in the best of our interest to define layers based on the GMP framework. To accomplish this task, we use some observations studied in Section 5.2. Let us define two GMP based layers.

The first is a fully connected layer and it has units that compute their outputs following Equation 5.1. We refer to this layer as \mathbb{I} -set descriptor- \mathbb{L} -fuzzy aggregation layer. We will focus on the \mathbb{I} -set descriptor- \mathbb{L} -fuzzy aggregation layer that makes use of the pair $P = (\sum, [\phi_\times])$ with excitatory output responses. We call Sum-Dot or SD layer, for short, whenever the pair $P = (\sum, [\phi_\times])$ is used. The computation of each unit of an SD layer is given in Equation 5.8.

The second layer has exactly one node per class and computes a operation similar to Equation 5.3, which implies that each unit performs the union or intersection strategy.

Sum-Dot GMP Layer

The **Sum-Dot GMP Layer** (SD) is a feedforward fully connected layer where each node computes a forward pass function $f_i^\ell : \mathbb{R}^n \rightarrow \mathbb{R}$, where $n = |F_{\ell-1}|$ as follows:

$$f_i^\ell(\mathbf{x}) = \sum_{j=1}^n (x_j - \underline{w}_j^i) \cdot (x_j - \overline{w}_j^i) \quad (5.8)$$

where \mathbf{x} is the output of the previous layer. The values of gradients and the backward pass functions are computed following the derivatives given in Equations 2.75, 2.76 and 2.77. Note that this layer is fully connected.

The learnable parameters of a unit of computation at SD layers are intervals $[\underline{\mathbf{w}}^j, \overline{\mathbf{w}}^j]$ in

$\mathbb{I}_{\mathbb{R}}^n$ which are initialized as random hyperboxes. First, each $\underline{\mathbf{w}}_i^j$ is initialized with a random value that considers a small Gaussian with mean 0 and variance according to the number of inputs and outputs of the layer. Second, an ϵ_i^j is initialized using the same strategy. Then, we compute $\overline{\mathbf{w}}^j = \underline{\mathbf{w}}_i^j + |\epsilon_i^j|$. This strategy initializes the parameters as small intervals with different lengths.

Observation #1: If the number of classes $|L|$ and nodes $|F_\ell|$ at SD layers are the same, the class aggregation layer becomes unnecessary. Therefore, not every SD layer has a class aggregation attached.

Observation #2: To have more than one SD unit of computation per class, it is necessary to do a partition of the data. This way, the training data that belong to a certain is partitioned in smaller subsets, and each one of these subsets is assigned to a one Sum-Dot unit of computation. This solution is similar to the the solution proposed in (ZAMORA; SOSSA, 2017). In this case, the training by SGD can be considered a tuning of weights for other algorithms. Nevertheless, this process is not the focus of this thesis and we will work without the class aggregation layer. Hence, we work with one unit of computation per class.

Class Aggregation GMP Layer

The **class aggregation GMP layer** F_ℓ computes the forward pass for each unit of computation as follows:

$$f_i^\ell(\mathbf{x}) = \bigvee_{k \in K_i} x_k, \quad \text{for } i = 1, \dots, |L| \quad (5.9)$$

where \mathbf{x} is the input coming from the previous layer, $|L|$ is the number of available labels, $K_i \neq \emptyset$, for $i \neq j$, $K_i \cap K_j = \emptyset$ is the set of units of computation that belong to class i . Therefore, the class aggregation GMP Layer acts as a router. The backward pass works in the opposite direction, hence, the gradients passed to the previous layer are zero on these places discarded at the forward pass. In this study we assume the use of Sum Dot as the unit of computation of GMP layers, which we had proven is differentiable.

Observation #3: In order for these layers to become useful architectures in the deep learning context, a proper structure must be defined. Hence, the next step is to define some hybrid DL/morphological models and its training by SGD.

5.4 The Sum-Dot Model

In this section we present the simplest model that involves the layers defined in Section 5.3.

This model can be considered a model with at least four layers that makes use of a Sum-Dot layer to create the decision surface. The first layer F_1 is the input layer that can perform a

zero-pad normalization, a gaussian filter or an identity function, as specified in Section 3.4. The second is a GMP Layer F_2 that computes some \mathbb{I} -set descriptor and an aggregation at each sub-module. We focus on the Sum-Dot pair, which is differentiable. The third layer is a class aggregation and F_3 that computes a union or intersection wise strategy. Note that, layer F_3 is optional if the number of nodes of layer F_2 are already equal to the number of classes. Next, we recommend a soft-max layer F_4 that makes use of a cross entropy loss function at layer F_5 . Note also that, layers F_4 or F_5 can be replaced by any other mechanism that computes a loss function, such as those studied in 3.2. Nevertheless, we stated before that we will focus our descriptions on Sum-Dot and softmax-cross entropy, since softmax-cross entropy is a well succeeded loss function in classification.

The experiments of Chapter 6 assume this configuration. Besides, we assume that layer F_3 has exactly one node by class and, therefore, neither the union nor the intersection strategy is required in the simulations.

Next, we present a backpropagation numerical example designed to understand the behavior of this model. Let us assume a binary problem with $L = \{c^1, \dots, c^m\}$ classes and a training dataset $\mathcal{T} = \{(\mathbf{x}^1, y^1), \dots, (\mathbf{x}^k, y^k)\}$ such that the observations are vectors $\mathbf{x}^i \in \mathbb{R}^n$. Without loss of generality, let the input layer perform an identity. Assume an SD layer with two nodes so that each node represents one class. Hence, the class aggregation can be omitted. Therefore, the learnable parameters are $[\underline{\mathbf{w}}^s, \overline{\mathbf{w}}^s]$, for $s = 1, 2$. Finally, assume the softmax-layer and CE loss function. As aforementioned, the gradients from the softmax layer to the SD layer are given by equation 3.34, Therefore, the partial derivatives with respect to \underline{w}_i^s and \overline{w}_i^s are given by:

$$\begin{aligned} \frac{\partial L}{\partial \underline{w}_i^s} &= (x_i - \overline{w}_i^s) \cdot \sum_{j=1}^m (o_j - y_j) \\ \frac{\partial L}{\partial \overline{w}_i^s} &= (x_i - \underline{w}_i^s) \cdot \sum_{j=1}^m (o_j - y_j) \\ \frac{\partial L}{\partial x_i} &= \sum_{j=1}^m (o_j - y_j) \cdot (\underline{w}_i^j + \overline{w}_i^j - 2x_i) \end{aligned}$$

Example 6. Consider the same model presented above and an observation $\mathbf{x} = [0.2, 0.5, 0.8]$ that belong to the class c_1 . Assume that the weights of $\underline{\mathbf{w}}^1 = [0.3, 0.4, 0.6]$, $\overline{\mathbf{w}}^1 = [0.4, 0.6, 0.7]$, $\underline{\mathbf{w}}^2 = [0, 0.5, 0.7]$ and $\overline{\mathbf{w}}^2 = [0.2, 0.6, 0.7]$. Thus, the outputs at the SD layer are:

$$\mathbf{y}_1 = [-0.03, -0.01]^T;$$

and therefore, the softmax layer outputs are:

$$\mathbf{y}_2 = [0.495, 0.505]^T;$$

which implies that the observation was misclassified.

In this example, the gradient values computed at the softmax layer are:

$$\mathbf{gd}_1 = [-0.505, 0.505]^T;$$

And the gradients for the learnable parameters at the SD layer are:

$$\begin{aligned}\nabla_{\underline{\mathbf{w}}}^1 &= [0.1010, 0.0505, -0.0505]^T, & \nabla_{\underline{\mathbf{w}}}^2 &= [0, -0.0505, 0.0505]^T \\ \nabla_{\overline{\mathbf{w}}}^1 &= [0.0505, -0.0505, -0.1010]^T, & \nabla_{\overline{\mathbf{w}}}^2 &= [0.1010, 0, 0.0505]^T\end{aligned}$$

Using a learning step $\gamma = 0.5$, the new values of parameters at this iteration are:

$$\underline{\mathbf{w}}^1 = [0.3, 0.4, 0.6]^T - 0.05\nabla_{\underline{\mathbf{w}}}^1 = [0.2495, 0.3748, 0.6252]^T \quad (5.10)$$

$$\overline{\mathbf{w}}^1 = [0.4, 0.6, 0.7]^T - 0.05\nabla_{\overline{\mathbf{w}}}^1 = [0.3748, 0.6252, 0.7505]^T \quad (5.11)$$

$$\underline{\mathbf{w}}^2 = [0, 0.5, 0.7]^T - 0.05\nabla_{\underline{\mathbf{w}}}^2 = [0, 0.5252, 0.6748]^T \quad (5.12)$$

$$\overline{\mathbf{w}}^2 = [0.2, 0.6, 0.7]^T - 0.05\nabla_{\overline{\mathbf{w}}}^2 = [0.1495, 0.6000, 0.6748]^T \quad (5.13)$$

And with these new parameters, the new output is:

$$\mathbf{y}_1 = [-0.0016, -0.0283]^T$$

and therefore, the softmax layer output is:

$$\mathbf{y}_2 = [0.5067, 0.4933]$$

which implies that the observation \mathbf{x} was correctly classified.

In this simple example, we can analyze how the parameters vary with respect to an input. Considering the class of \mathbf{x} is c^1 , we saw that for i -th dimension, values $\underline{w}_i^1, \overline{w}_i^1$ becomes smaller whenever $x_i \leq \underline{w}_i^1$. If we assume $\underline{w}_i^1 \leq x_i \overline{w}_i^1$ then \underline{w}_i^1 become smaller and \overline{w}_i^1 larger. Whenever $\overline{w}_i^1 \leq x_i$, the values $\underline{w}_i^1, \overline{w}_i^1$ become larger.

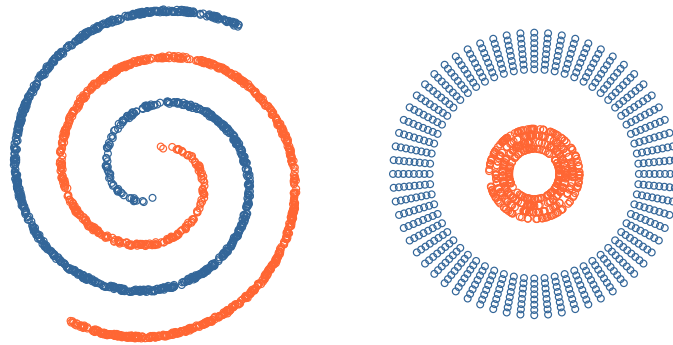
On the other hand, $\underline{w}_i^2, \overline{w}_i^2$ become larger whenever $x_i \leq \underline{w}_i^2$. If we assume $\underline{w}_i^2 \leq x_i \overline{w}_i^2$ then \underline{w}_i^2 becomes larger and \overline{w}_i^2 smaller. Whenever $\overline{w}_i^2 \leq x_i$, values $\underline{w}_i^2, \overline{w}_i^2$ become smaller.

The interpretation of this observation is simple: whenever \mathbf{x} belongs to the s -th unit of computation, the sequence of intervals $[\underline{\mathbf{w}}^s, \overline{\mathbf{w}}^s]$ generated by the SGD algorithm will try to enclose \mathbf{x} . For the other units, the sequence will try to move away intervals from the observation. In this model, the gradient values w.r.t. inputs \mathbf{x} are not used, hence, the SD Layer does not propagate their errors to any previous layer. Nevertheless, this perspective can entail some problems. Figure 25 illustrates binary classification examples, in which the separation using a single unit of computation per class is not possible. In that sense, constructive algorithms presented in Section 4.4 propose to build units of computation incrementally. However, we will propose a deeper topology in which other transformations are performed before the computation of SD layer units to deal with these problems.

5.5 FC-SD Models

The second model proposed here propagates the gradient through the SD to previous FC layers. In this hybrid model, one or a set of FC layers perform pre-processing

Figure 25 – Spiral and Included Datasets



Blue dots and red dots, represent observations of class C_0 and C_1 respectively. These two datasets can't be separated using a single unit of computation by class with the SD-Softmax model.

of the data. In this hybrid method non-linear FC layers are followed by activation layers. Then an SD layer performs classification.

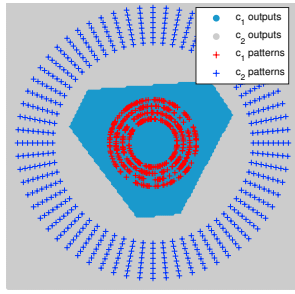
FC-SD models tested in this thesis have at least seven layers. Next we specify a minimalist model used here.

1. The first layer is the input layer.
2. The second layer is a weighted fully connected layer.
3. The third is a batch normalization layer. This layer is optional. However, since SD parameters are intervals, the covariance shift may cause problems in earlier layers.
4. The fourth layer is an activation function, we recommend a linear rectifier.
5. Items from 2 to 4 can be applied in cascade more than once. (optional)
6. Next layer is an SD layer with a number of unit of computation equal to the number of classes.
7. The last two layers compute a soft-max and a cross entropy loss function at training stage.

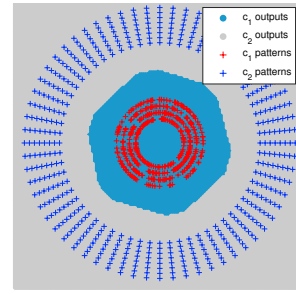
Observation #1: Since this model relies on a set of linear transformations, which extract features from an input and a SD layer that performs classification, we will refer to this model as FC-SD.

Observation #2: Although the batch normalization layer is optional, it improves the convergence and diminishes the oscillations of the BP algorithm, mainly at early stages. In order to check if this model has the capacity to deal with the datasets illustrated in Figure 25, we perform a simulation. In this simulation, the number of nodes at the FC

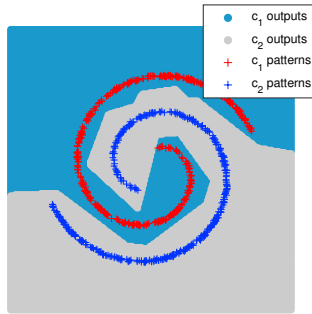
Figure 26 – FC-SD-Softmax Model Test on Spiral and Included Datasets



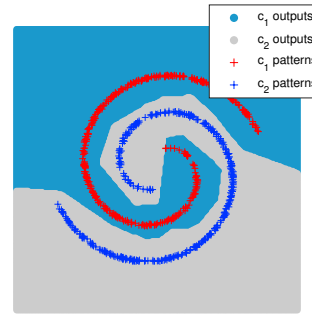
(a) Class separation of dataset using three nodes at fully connected layer and two nodes at SD.



(b) Class separation of included dataset using six nodes at fully connected layer and two nodes at SD.



(c) Class separation of spiral dataset using eight nodes at fully connected layer and two nodes at SD.



(d) Class separation of included dataset using fifteen nodes at fully connected layer and two nodes at SD.

layer is varying (cf. Figure 26).

We trained the FC-SD models using the entire datasets. Then, we plotted the resulting decision surface generated by model. The reader can notice that a perfect class separation was achieved in these cases. Nevertheless, in these simple examples we do not prove the generalization capability of the model. In fact, it is only useful to check that the FC-SD-SM model computes much more complex surfaces than the SD-SM model whenever a single unit of computation per class is considered in the SD layer.

5.6 Conv-SD Models

The last hybrid model presented here uses GMP layers to perform classification and a convolutional part that is responsible for feature extraction. This architecture has at least 8 layers.

1. The first layer is the input layer and could perform some normalization function or an identity.
2. The second layer is a convolutional layer. The configuration for convolution filters,

stride and padding is dependent of the problem. It is followed by some activation functions. In general, as activation function BN+ReLU layers are recommended.

3. Next, a pooling layer computes a downsampling operation.
4. The layers described in items 2 and 3 can be applied in cascade more than once (optional).
5. Note that, last layers belong to the convolution part. After some cascades of convolution and downsampling, we can consider the FC-SD model to perform the classification part. Therefore, an architecture based in the FC-SD model presented before can be attached to the convolutional part.
6. The last layers are concerned with generating a loss function. We recommend a softmax and to use the cross entropy loss function at the training stage.

Observation #1 Since the last model makes use of convolution in the early layers, its use is restricted to analyze datasets with spatial information.

Observation #2 The convolutional part of a Conv-SD model, may vary depending on the problem.

Observation #3 The SD part of a Conv-SD model differs from a traditional CNN, or Conv-FC to be more exact, in the next to last layer. Instead of using an FC layer with the number of nodes equal to the number of classes, Conv-SD models make use of a GMP layer. Specifically, we use an SD layer.

6 Some Experimental Results

Don't compete! — competition is always injurious to the species, and you have plenty of resources to avoid it!

Peter Kropotkin, *Mutual Aid: A Factor of Evolution*

The purpose of this chapter is to compare the classification performance of the models presented in this work with some well-known related classifiers from the literature. It is important to mention that the classification models proposed in Sections 5.4 and 5.5 do not depend on any spatial structure. In general, these classifiers work on any classification dataset. On the other hand, the models proposed in Section 5.6, which make use of convolutional operators, are designed to work with spatial structures such as images. Hence, datasets to test Conv-SD are image datasets. Therefore, two types of datasets must be considered to test the models presented in Chapter 5.

The first set of experiments concerns some of the datasets used in (SUSSNER; CAMPIOTTI, 2020) that were drawn from the UCI Machine Learning Repository (ASUNCION; NEWMAN, 2007). Besides traditional classification models, some lattice based and morphological classifiers presented in Chapter 4 were selected to compare the performance of the models proposed in Sections 5.4 and 5.5. A second experiment was made using a subset of the MNIST digits database (LECUN et al., 1998).

The other experiments are concerned with image classification. The image datasets for classification selected are the Robot Visual Self-Localization (COUNTRY.,), the MNIST digits database (LECUN; CORTES, 2010) and the CIFAR-10 database (KRIZHEVSKY; HINTON et al., 2009). Here, we test the models proposed in Chapter 5. Besides some classifiers studied in Chapter 3, we selected some classifiers specialized in image processing to compare results.

6.1 General Classification Problem Datasets

In this section we present an analysis of the SD and FC-SD models on ten different datasets. The purpose is to compare the classification performance with some related models presented in Chapter 4, namely:

- MP/CL (SUSSNER; ESMI, 2009b; SUSSNER; ESMI, 2011).

- DMNN-SGD (ZAMORA; SOSSA, 2017).
- DELP (ARAÚJO; OLIVEIRA; MEIRA, 2012).
- HMLP-EL (SUSSNER; CAMPIOTTI, 2020).
- MLP-EL (HUANG; ZHU; SIEW, 2006; HUANG, 2014).
- MLP-BP (HINTON; SRIVASTAVA; SWERSKY, 2012).

Besides morphological classifiers, other classifiers were considered, namely:

- Nearest Neighbors (ALTMAN, 1992).
- Naive Bayes (HASTIE; TIBSHIRANI; FRIEDMAN, 2009).
- Support Vector Machine (CORTES; VAPNIK, 1995).
- Bagging Random Forest (PRASAD; IVERSON; LIAW, 2006).
- XG-Boost with softmax (CHEN et al., 2015; CHEN; GUESTRIN, 2016).

The simulations were conducted on eleven datasets taken from UCI Machine Learning Repository. ADAM, and learning decay schedule were considered for the following models: SD, FC-SD and MLP-BP. The hyperparameters used to train these models were a Squared decay factor of 0.999, a momentum of 0.95 for adaptive momentum and a learning rate decay factor of 0.5. The decay schedule performs the decay factor of the learning rate 3 times, considering the max number of epochs. For the FC-SD and the MLP-BP models fully connected weights and L2 regularization with a multiplier 0.0001 were applied. In general, for SGD related models these parameters will be used in most of the experiments performed in this chapter. The number of epochs, learning rates, nodes in the hidden layers and other parameters were computed using a random search taking into account mean accuracy.

The Iris Dataset

The Iris flower dataset or Iris Dataset (IRD), for shortly, is a dataset with multiple measures introduced by Ronald Fisher in (FISHER, 1936). The data consist of 150 well balanced observations from three different species of Iris (*Iris Setosa*, *Iris Virginica* and *Iris Versicolor*). The features of each observation are: length and width of sepals and petals, in centimeters. Hence is a dataset with four features. This dataset became a typical test case for classification.

Table 5 – Hyperparameters computed for the datasets selected from UCI.

	Names	IRD	PAD	LED	WID	BCD	IQD	CAD	DED	LCD	LYD	SMD
SD-SM	Epochs	60	80	60	30	80	100	50	80	80	50	80
	LR	0.05	0.010	0.010	0.050	0.050	0.01	0.05	0.05	0.01	0.080	0.010
	Batches	16	128	256	9	8	32	15	16	8	32	64
FC-SD-SM	Epochs	60	80	80	80	50	50	60	60	50	40	80
	LR	0.006	0.003	0.01	0.01	0.02	0.05	0.001	0.002	0.02	0.03	0.02
	FC Layer	35	100	150	30	200	100	60	50	100	30	100
XGB	Batches	16	128	60	12	32	32	50	30	20	8	256
	MaxDepth	5	4	8	5	3	5	5	4	6	5	8
	LR	0.1	0.05	0.2	0.1	0.1	0.01	0.2	0.1	0.08	0.09	0.2
RF-BAG	Cycles	10	14	11	34	13	198	84	12	42	11	10
	Split Crit	gdi	gdi	t-ing	gdi	gdi	dev	gdi	gdi	dev	dev	t-ing
	Min Leaf	2	5	2	1	3	2	1	3	2	2	1
MP/CL	Max Splits	88	471	7782	4	30	278	352	14	7	57	257
	Hyperparameters not required											
	LR	0.03	0.02	0.01	0.05	0.004	0.007	0.01	0.0001	0.01	0.05	0.04
TSGD-MP	Method	K++	nHpC	K++	HpC	HpC	HpC	K-means	HpC	nHpC	nHpC	nHpC
	Hyperboxes	4	2	1	-	-	-	2	0	2	8	4
DELP	Modules	60	150	250	50	150	200	200	80	150	120	200
	LR	0.01	0.01	0.01	0.01	0.005	0.01	0.01	0.01	0.1	0.01	0.01
HMPL-EL	Linear Nodes	60	132	2000	3000	800	1000	500	200	100	200	1100
	Morphs Mods	200	300	2300	2900	1000	1200	800	600	1000	500	1500
MLP-EL	Linear Reg	0.0010	0.0200	0.1	1	1	1	1	0.9	0.1000	0.8	0.9
	Hidden	200	400	5000	3000	3500	1500	900	350	5000	5000	2000
MLP-BP	Linear Reg	1.1	0.01	0.001	1	1	1	1	0.5	.01	.5	1
	Batches	16	128	60	12	32	32	50	30	20	8	256
KNN	LR	0.008	0.001	0.010	0.01	0.01	0.05	0.001	0.004	0.01	0.02	0.02
	H. FC Layer	50	110	130	50	180	80	60	90	100	30	100
	Epochs	60	80	80	80	50	50	60	60	50	40	80
Naive Bayes	Neighs.	3	5	1	1	3	1	11	8	1	14	1
	Distance	Euc	Cos	Euc	CB	CB	CB	Cos	CB	SP	CB	Corr
SVM	Width	0.15	0.34	0.44	0.11	0.22	0.11	31.07	0.21	1.1	0.43	1.8
	Kernel	N	EP	N	N	EP	N	EP	N	EP	TR	EP
	Constraint	45.20	1.61	0.034	477.16	0.58	999.06	0.0040	0.34	9.8543	6.47	263
SVM	Kernel Scale	1.81	-	-	285.01	-	-	-	-	-	-	-
	Polynomials	-	3	-	-	-	-	-	-	3	-	3
	Coding	onevsall	onevsall	onevsall	onevsone	onevsone	onevsone	onevsall	onevsall	onevsone	onevsall	onevsone
Method	G	P	L	G	L	L	L	L	L	P	L	P

We considered eleven datasets that belong to the UCI Machine Learning Repository (ASUNCION; NEWMAN, 2007) and thirteen different classifiers.

The Pages Dataset

The Pages Dataset (PAD) was originally created by Donato Malerba ([ESPOSITO; MALERBA; SEMERARO, 1994](#); [MALERBA; ESPOSITO; SEMERARO, 1996](#)) The dataset has 5,473 observation taken from 54 documents. Each observation has 10 attributes that concern one block property and all these attributes are numeric. The dataset has 5 classes and it was designed to accomplish a task called document classification. Document classification aims at identifying the membership class of a document, provided that the user is able to define a set of classes that are relevant for his specific application ([ESPOSITO; MALERBA; SEMERARO, 1994](#); [ASUNCION; NEWMAN, 2007](#)).

The Letter Dataset

The Letter Dataset (LED) was originally created by David J. Slate and was designed to identify each of a large number of black-and-white rectangular pixel displays as one of the 26 capital letters in the English alphabet. Character images were based on 20 different fonts and each letter within these 20 fonts was randomly distorted to produce a file of 20,000 unique observations. Each observation was converted into 16 primitive numerical attributes (statistical moments and edge counts) which were then scaled to fit into a range of integer values from 0 through 15. We train on the first 16,000 items and then we used the resulting model to predict the letter category for the remaining 4,000 observations ([FREY; SLATE, 1991](#); [ASUNCION; NEWMAN, 2007](#)).

The Wine Dataset

Data contained in the Wine Dataset (WID) is a result of chemical analysis of wines grown in the same region of Italy and derived from three different cultivars. Hence the data have 13 features with 3 classes defined by the 3 cultivars ([AEBERHARD; COOMANS; VEL, 1994](#)). The analysis of the wine determined quantities for 13 properties found in each of the three types of wines. Note that all attributes are continuous ([FORINA et al., 1988](#)).

The Cancer Dataset

The Breast Cancer Diagnostic Dataset (BCD) is a binary classification dataset and was initially proposed by W. Wolberg. The process to get the features involves an image digitalization of a fine needle aspirate (FNA) of a breast mass ([STREET; WOLBERG; MANGASARIAN, 1993](#)). The attributes of each observation describe characteristics of the cell nuclei present in the image and there are ten features that describe each sample. The task to be accomplished is automated cancer diagnosis using these characteristic taken from the cell nuclei ([DEMIR; YENER, 2005](#)).

The Ionosphere Quality Dataset

The Ionosphere Quality Dataset (IQD) was collected by the Space Physics Group of The Johns Hopkins University Applied Physics Laboratory. The data was collected by a system in Goose Bay, Labrador and consists of a phased array of 16 high-frequency antennas with a total transmitted power on the order of 6.4 kilowatts. Broadly speaking, the radar operates by transmitting a multi-pulse pattern to the ionosphere. Then, the receiver is turned on between pulses, and the target velocity is determined by measuring the phase shift of the returns. The targets were free electrons in the ionosphere. There were 17 pulse numbers for the Goose Bay system. Instances in this database are described by 2 attributes per pulse number, corresponding to the complex values returned by the function resulting from the complex electromagnetic signal. "Good" radar returns are those showing evidence of some type of structure in the ionosphere. "Bad" returns are those that do not; their signals pass through the ionosphere. Ionosphere dataset is a binary classification problem with 34 continuous measures for each observation (SIGILLITO et al., 1989). Nevertheless, since the second variable of the dataset has a 0 standard deviation, it was removed from the dataset. Hence, experiments were conducted using 33 characteristics for each observation.

The Arrhythmia Dataset

The Cardiac Arrhythmia Dataset (CAD) contains 279 attributes, 206 of which are linear valued and the rest are nominal. The aim is to distinguish between the presence and absence of cardiac arrhythmia and to classify it in one of the 16 groups. Class 01 refers to 'normal' ECG classes 02 to 15 refers to different classes of arrhythmia and class 16 refers to the rest of unclassified ones (GUVENIR et al., 1997). It is important to mention that this dataset contains missing values. Attributes with missing values were ignored. Besides, some attributes with redundant data were not considered.

The Dermatology Dataset

The differential diagnosis of erythemato-squamous diseases is a difficult problem in dermatology. They all share the clinical features of erythema and scaling, with very few differences. The diseases in this group are psoriasis, seboreic dermatitis, lichen planus, pityriasis rosea, chronic dermatitis and pityriasis rubra pilaris (GÜVENIR; DEMİRÖZ; ILTER, 1998). The Dermatology Dataset (DED) is a classification dataset that considers those diseases. Hence, the dermatology dataset is a multi-class problem with six classes. The dataset contains 13 clinical and 22 histopathological features for each individual observation. The clinical features considered include family history and symptomatic characteristic observed in individuals. To collect histopathological features, analysis of the

samples under a microscope were done. The number of observations of the dataset is 366, which means that 366 individuals with erythematous-squamous diseases were considered.

The Lung Cancer Dataset

The Lung Cancer Dataset (LCD) describes 3 types of pathological lung cancers. All the features are nominal and they take values from 0 to 3. We selected 54 of the 56 original features. In this case, we excluded two features with missing values. The dataset describes 3 types of pathological lung cancers and it was introduced by Hong, and Yang in 1991 (HONG; YANG, 1991).

The Lymphography Dataset

the Lymphography Dataset (LYD) was developed from the University Medical Centre, Institute of Oncology in Yugoslavia (CESTNIK; KONONENKO; BRATKO, 1987). This dataset has repeatedly appeared in the machine learning literature and it has been widely studied (HUTTER; ZAFFALON, 2005). The idea is to classify the lymphography and detect normal, metastases, malign or fibrosis lymph. The LYD is a classification dataset with 4 classes and 148 observations.

The Segmentation Dataset

The Segmentation Dataset (SMD) was created by the Vision Group in the University of Massachusetts and the observations were taken from a database of 7 outdoor images. To create the instances the images were handsegmented and all the pixel were processed. Then, each observation represents a 3×3 regions with 19 features (BAGIROV et al., 2003).

Other Considerations

Each dataset was randomly partitioned, the training dataset consists of 80% of the data and the test dataset consists of 20%. The set of hyper-parameters for the models was computed considering a random search and the best classification performance in the validation stage. In particular, we considered 20% of the original training dataset for validation purposes. We used 512 different random hyperparameter configurations. This is, in our simulations, the set of parameters tested in the random search for the SGD models has the following configuration: The number of epochs considered were $[5, 250] \in \mathbb{N}$, the initial learning rate has values in $[0.00001, 1]$ and the number elements of the batches has values in $\{8, 16, 32, 64, 128, 256\}$. In addition, for the models with hidden layers, we consider 1 to 1000 nodes in the random search. Therefore, 512 random combinations of

Table 6 – Classification Accuracies and F1-Scores in the Testing Stage.

	IRD	PAD	LED	WID	BCD	IQD	CAD	DED	LCD	LYD	SMD	AVG
SD-SM	96.67	100	76.78	97.14	93.27	91.43	56.81	97.95	83.33	90.50	95.19	89.01
	96.63	100	76.57	97.50	92.91	90.96	25.52	97.59	85.37	71.49	95.46	84.55
FC-SD-SM	96.67	100	96.40	97.14	95.58	93.14	70.22	97.12	80.67	90.21	97.49	92.24
	96.63	100	96.40	97.50	95.32	92.70	26.45	96.65	76.07	64.35	97.65	85.43
XGB	96.67	100	94.98	100	95.58	90	74.73	94.52	66.67	89.29	96.97	90.85
	96.63	100	94.97	100	95.29	89.23	32.73	93.35	66.67	50.99	97.12	83.36
SVM	96.67	100	94.83	97.14	97.35	87.14	70.33	97.26	66.67	89.66	95.45	90.23
	96.63	100	94.82	97.44	97.22	85.84	29.16	96.86	66.67	60.61	95.64	83.72
RF-BAG	96.33	100	92.73	98.00	95.93	93.43	72.75	96.16	59.33	87.96	96.97	89.96
	96.26	100	92.74	98.00	95.68	92.85	30.26	95.45	49.12	52.77	97.12	81.84
HMPL-EL	96.67	100	91.62	91.71	95.04	94.14	55.27	97.26	80	84.91	92.32	89.00
	96.63	100	91.65	91.87	94.78	93.60	22.73	96.80	81.63	52.75	92.58	83.18
DMNN-SG	96.67	100	44.93	94.29	92.92	90	39.56	94.52	50	71.43	68.61	76.63
	96.63	100	46.16	94.47	92.57	89.13	19.72	94.40	50	67.30	73.68	74.92
DELPH	96.67	100	83.57	71.43	61.95	94.29	53.85	97.26	80	85.71	73.16	81.62
	96.63	100	84.65	71.84	38.25	93.92	7.50	96.86	81.63	48.72	74.88	73.50
MPCL	96.67	99.82	61	94.29	95.58	90	51.65	87.67	50	65.52	91.56	80.34
	96.63	99.51	64.04	94.47	95.29	89.56	20.71	86.27	50	75.91	92.58	78.63
MLP-EL	96.67	100	95.39	90.57	93.81	92.43	51.21	96.85	78	88.89	92.64	88.77
	96.63	100	95.39	90.71	93.41	91.99	20.03	96.11	59.55	89.43	92.86	84.19
MLP-BP	96.67	100	93.49	92.86	93.81	93.57	58.24	97.26	66.67	87.93	93.83	88.57
	96.63	100	93.47	93.44	93.41	93.18	28.93	96.6	66.67	59.7	94.09	83.28
E-KNN	96.67	99.91	94.87	100	98.23	94.29	62.64	97.26	40	89.66	94.81	88.03
	96.63	99.75	94.83	100	98.14	93.92	25.08	96.34	22.22	61.26	94.95	80.28
NB	96.67	99.82	74.45	97.14	92.04	97.14	59.34	97.95	66.67	89.66	89.18	87.28
	96.63	99.50	75.13	97.50	91.60	96.89	24.12	97.59	66.67	92.70	89.63	84.36

The performance was evaluated using the accuracy and the macro average F1-score measurements (ASCH, 2013). For each model the first row represents accuracy and the second row the macro average F1-score. In the last column the average (AVG) of both measures for each model is presented.

elements of these sets of parameters are tested in the validation set, the set of hyper-parameters with best accuracy is selected to train the model. For the SVM configuration gaussian (G) linear (L) and polynomial (P) kernels were considered in the random search. Besides, the onevsall and onevsone strategies for coding were selected in the validation stage together with the constraint value. For the gaussian case the kernel scale was also considered. In the case of the Naive Bayes Epanechnikov (EP), triangular (T) and normal (N) kernels were considered in the validation stage (FRIEDMAN; HASTIE; TIBSHIRANI, 2001; BÜHLMANN; GEER, 2011). For the KNN ensemble, we considered the following eleven distances: Cityblock (CB), Chebychev, Correlation (Corr), Cosine (Cos), euclidean (Euc), Hamming, Jaccard, Mahalanobis, Minkowski, seucclidean and spearman (SP). It is important to mention that Chebychev, Hamming, Jaccard, Mahalanobis, Minkowski and seucclidean were not selected by the random search for any dataset. In the particular case of MP/CL there are no hyper-parameters to compute. Therefore, the random search was omitted in this case.

For the bagging random forest (RF-BAG), the split criterion, the number of cycles, the max depth and the min leaf size were considered in the validation stage (STROBL; MALLEY;

TUTZ, 2009). The values for split criterion used in this case were gdi and deviance (LOH; SHIH, 1997; LOH, 2011). In the case of XGBoost (XGB) the max depth and the learning rate were considered in the validation stage. For the MLP-EL and HMPL-EL the linear regularizer and the number of hidden nodes were considered in the random search. In the case of the HMPL-EL the number of morphological nodes were also considered. Table 5 is a summary of the hyper-parameters values for each model and Table 6 shows the performance of the classifiers in testing phase.

Table 6 reveals competitive results of some morphological based classifiers. It was already pointed out in (SUSSNER; CAMPIOTTI, 2020) that the MP/CL model yields poor generalization accuracy when sparsely distributed datasets are considered. Particularly, the FC-SD model exhibited the best performance in four of the individual datasets and the SD-model exhibited the best accuracy in five datasets. Furthermore, the FC-SD model outperformed all the other models in terms of mean classification accuracy in the testing stage. The second place in terms of mean classification accuracy went to the XG-boost classifier, which achieved the best performance in four datasets. The ensemble of KNN showed competitive results and achieved the best result in three datasets. Note that, the SD model yields a poor accuracy in the Letter dataset, which is a dataset with 26 classes and 10 features. This poor generalization confirms our analysis on the SD and its problems to deal with datasets in which one class is included in another class (see Figure 25). Observe that we are using a single node for each class in the morphological layers, therefore to develop an algorithm that considers more nodes in the morphological layer could improve the results. However, it was mentioned that the strategy followed in this work is to rely on computing transformations before the morphological layer and to train the whole network using SGD-optimization.

We can also check that the F1-score reveals problems for almost all models in the classification of the CAD, LCD and LYD datasets. Especially in the CAD dataset. In that cases the F1-score was very low for the classification models. Nevertheless, for the LCD and LYD, the SD classifier performed a balanced classification with respect to the other classifiers. In the case of LYD, the SD achieved the best accuracy. Nevertheless, the NB classifier achieved the best F1-score. The DMNN-SG classifiers got the worst mean accuracy with a value of 76.63%. The best average accuracy was achieved by the FC-SD-SM with a value of 92.24%. The FC-SD-SM also achieved the best average F1-score with a value of 85.43%. The worst average F1-score was obtained by the DELP model with a value of 73.50%. In general, the results for the models presented in this work are competitive and it encourages us to do experiments on other datasets. In the following section we perform a simulation on a dataset with higher dimensionality.

6.2 MNIST1000 Digits Dataset

The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The digits have been size-normalized and centered in a fixed-size image in a 28x28 image by computing the center of mass of the pixels (LECUN et al., 1990; LECUN; CORTES, 2010).

This dataset is widely used to train, validate and test classifiers. There have been many efforts to achieve higher classification accuracies. In particular, some committees and hierarchical systems that combine the output of multiple classifiers have achieved excellent results. For example, the method presented in (KOWSARI et al., 2018) has been reported an error rate of 0.18%. In (ROMANUKE, 2016), a committee of 5 CNNs that makes use of data augmentation for training has an error rate of 0.21% (ROMANUKE, 2016). It is important to mention that a formal analysis of these types of ensembles becomes a hard task. Nevertheless, the common component between these results that have become the state of art for this classification problem is the use of deep learning and convolutional operations.

In order to perform a fair comparison between traditional Conv-FC and Conv-SD models, a strategy must be developed. As already mentioned, since the difference between a traditional Conv-FC model and the Conv-SD model presented here is in the classification part of the model, an option is to perform a comparison between models with the same topology except in the layer before softmax, where the FC layer of traditional CNNs can be replaced by an SD layer in the Conv-SDs models.

The next experiment considers some state of the art morphological classifiers and a reduced version of MNIST. In this case, we compare the results produced by some of the classifiers presented in Chapter 4 using a 1,200 samples subset of the entire MNIST, setting a 1,000 images for training/validation stages with a configuration of 80% for training and 20% for validation. The remaining 200 images were used for testing. A random search was performed to compute the hyperparameters of the models using 512 trials. Then, we selected the model set of hyperparameters with best accuracy. We will refer to this dataset as MNIST1000. Additionally, SD, FC-SD and the Conv-SD model introduced in Chapter 5 are included in this experiment. Table 7 shows the configuration computed in the validation stage and some state of the art morphological classifiers. The convolutional part of the Conv-SD model under consideration has the following configuration. The first layer is a convolutional layer with 12 filters followed by a BN and a ReLU layers. An average pooling layer performs downsampling in the fourth layer with size of 3 and stride 3. Another convolutional layer with 24 filters followed by BN and ReLU layers. Downsampling is applied once more using an average pooling with size of 3 and stride 3. Next, a FC layer with 128 nodes followed by BN and a ReLU layer. The morphological part follows the Conv part.

Additionally, a data-augmentation system was considered (PEREZ; WANG, 2017) for this

Table 7 – Accuracy and Model’s Configuration for MNIST-1000 Digits Dataset

	Parameters	Values	Training Accuracy	Test Accuracy
HMLP-EL	Linear	5000		
	Morphs	5000	100	94.00
	Reg	0.1		
MLNN	Morphs	100	100	88.29
	LR	0.001		
DMNN	Method	HpC	80.08	65.55
	LR	0.001		
DELP	Modules	60	95.55	84.75
MLP-EL	Linear	830	100	93.00
	Reg	0.001		
SD	Epochs	80		
	LR	0.01	100	85.35
	Batches	64		
	Method	ADAM		
FC-SD	Epochs	30		
	FC Nodes	64		
	LR	0.001	100	92.50
	Batches	64		
	Method	SGDM		
Conv-SD	L2-Reg	0.5		
	Epochs	25		
	LR	0.001	100	96.50
	Batches	64		
	Method	SGDM		
DA-SD	Pooling	Average		
	Epochs	80		
	LR	0.01	85.20	87.20
	Batches	64		
DA-FC-SD	Method	ADAM		
	Epochs	150		
	FC Nodes	128		
	LR	0.08	99.40	95.50
	Batches	64		
DA-Conv-SD	Method	SGDM		
	Epochs	150		
	LR	0.08	99.70	98.85
	Batches	64		
	Method	SGDM		
	Pooling	Average		

experiment. The data-augmentation adopted here considers scaling, rotation, translation, X-shear and Y-shear transformations. Instead of the off-line strategy reported in (ROMANUKE, 2016), in which 560,000 images are created using various transformations and 180,000 are selected to carry out a traditional BP training with the augmented dataset, we use an online strategy, which is more dynamic. Note that in one epoch the entire dataset is processed. An online strategy is applied to the aforementioned transformations to obtain new images in each epoch and for each mini-batch. This way at each epoch an augmentation on the fly creates new datasets of 60,000 in each epoch for each mini-batch. Online

strategies are more suitable for larger datasets, because potentially explosive increase in storage when off-line strategies are used is avoided (SHORTEN; KHOSHGOFTAAR, 2019). A second test of the SD, FC-SD and Conv-SD is performed using the same architectures but using data-augmentation. In this case, we use the acronyms DA-SD, DA-FC-SD, and DA-Conv-SD, respectively. This data-augmentation strategy will be assumed in most of the experiments presented here.

As expected, both Conv-SD methods achieved the highest classification accuracies on the MNIST1000 dataset among the models we considered. This is understandable since Conv-SD methods make use of the spatial information of the images. Also, the result reveals a considerable increase of accuracy in those methods in which data-augmentation was applied. The third best accuracy was obtained by DA-FC-SD that outperformed HLMP and FC-SD models. This fact proves the importance of data augmentation in FC-SD models. It is important to mention, in this case extreme learning machine strategies were successful in reaching better local minima than other non-convolutional SGD-based classifiers when data-augmentation was not applied. Nevertheless, we have to consider that the number of morphological and linear nodes of the HMLP-EL to achieve a 94% accuracy was 10,000. Besides, to train the HMLP-EL model it is necessary to perform the computation of matrix inverses or pseudo-inverses, which makes virtually impossible to use the HMLP-EL on the original MNIST or other larger datasets.

6.3 MNIST Digits Dataset

The next experiment considers the complete MNIST digit dataset. A Conv-SD and a Conv-FC model with 26 layers were tested. The convolutional part of both, Conv-SD and Conv-FC topology is:

The first layer is convolutional with 32 filter 3×3 sized, followed by BN and ReLu Layers. Next, we repeat the same configuration of the first three layers. In the seventh layer we perform downsampling, this time using a convolutional layer with 32 filters 5×5 sized and stride size of 2, no padding is applied. The layers 8 and 9 are a BN and ReLu layer, respectively. A dropout operation is applied in the DO layer 10. Next, a convolutional layer with 64 filters, each one 3×3 sized, followed by BN and ReLu Layers. Layers 14 to 16 have the same configuration of layers 11 to 13. Next, a downsampling is applied using another convolution layer with 64 filters, size of 5, stride 2 and without padding. The layers 18 and 19 are a BN and ReLu layer, respectively. The twentieth layer is a DO layer with 0.4 dropout probability. Next, a FC layer with 128 nodes followed by a BN and a ReLu layer constitute the last three layers of the Conv part. As in the other experiments the difference between Conv-SD and Conv-FC is the use of an SD and a FC layer in the ante-penultimate layer, respectively. For both models, a softmax and a cross entropy layer are used to train the model.

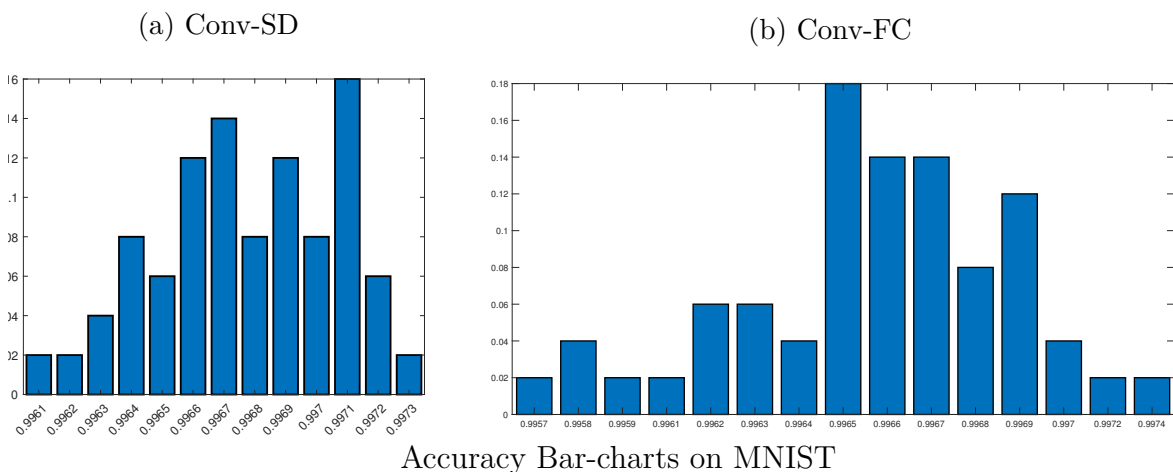
One of the biggest problems with SGD based algorithms is the quality of the local minimum reached. Due to the random initialization of the parameters, data-augmentation and the stochastic nature of the algorithm, among other factors; the local minimum reached may be different each time the algorithm is executed. Therefore, qualities of the extrapolation depend of the local minimum reached. Our first analysis is exactly that, to measure the quality of local minimums reached by the aforementioned model. For each model, the training was performed 100 times, generating 100 different models. We present bar-charts of their accuracy and an analysis of test observation that were misclassified by several different models. Then, we show the set of test observations that were misclassified at least, 50 times. The second analysis is performed on the classifiers with best accuracy per architecture. We use a confusion matrix to analyze the performance achieved in test stage. The following hyperparameters were chosen to train the model: The initial learning rate is 0.001, the learn drop period is 5, the learn drop factor is 0.5, the maximum number of epochs is 40, the mini batch size is 256 and the SGD method used is Adam. We perform an individual analysis of both models. In addition, we test some committee configurations, in order to improve the classification.

Conv-SD: Results on MNIST

Next, we present an analysis of the results obtained by the Conv-SD model in the MNIST experiment. Figure 27a illustrates the accuracy obtained in a bar-chart. The minimum value achieved was 0.9961%, which means that the maximal number of misclassified images is 39. The maximum value obtained was 0.9973%. The mean accuracy was 99.68% with a standard deviation of 0.0003. Besides, accuracy of 99.71% has the highest number of occurrences.

We analyzed which images were misclassified in the simulations. Figure 28 illustrates in a

Figure 27 – Accuracies on MNIST dataset

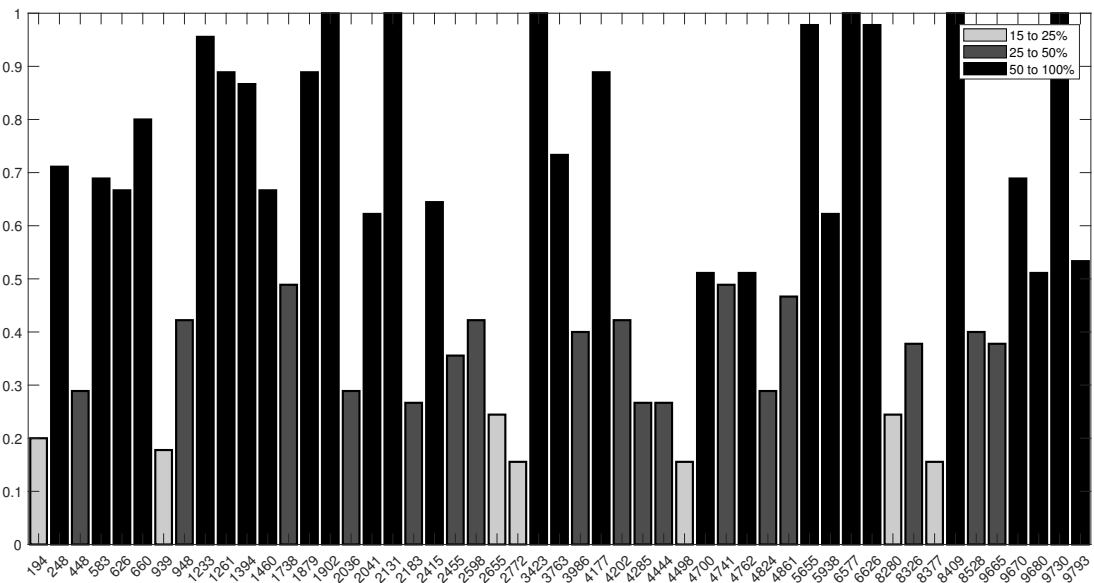


bar-chart these digits that were misclassified with different number of occurrences. The

number of images that were misclassified at least one time by some of the 100 classifiers, is 108. The number of digits misclassified at least by 15% and at most 25% of the classifiers is 8. The misclassified digits with more than 25% and at most 50% are 20. The number of images misclassified by more than 50% of the classifiers was 23. The set of digits misclassified by more than 50% of the classifiers is illustrated in Figure 30a. We refer to them as the hard digits set for the Conv-SD architecture.

As aforementioned, among the 100 Conv-SD classifiers, the best accuracy reached was

Figure 28 – Conv-SD Bar-chart Common Errors for MNIST.

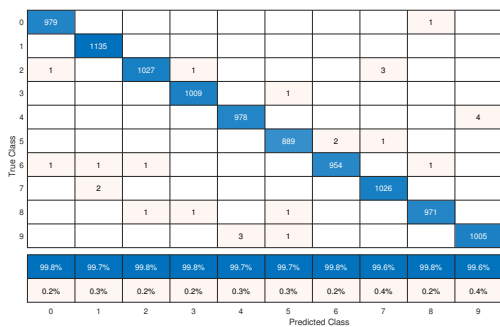


0.9973%, which means that it misclassified 27 of the 10000 test images. The confusion matrix of the classification performed can be visualized in Figure 29a.

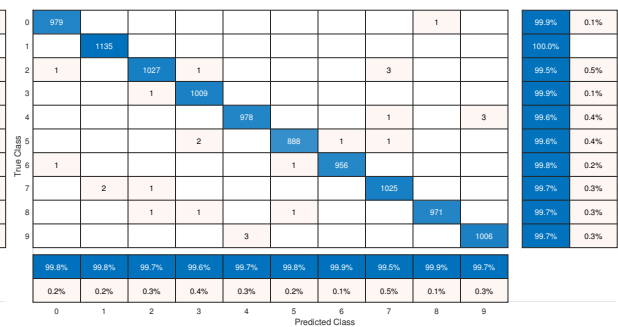
In general, the results are satisfactory. Nevertheless, note that the set of hard digits is

Figure 29 – MNIST Confusion Matrices

(a) Conv-SD Confusion Matrix for MNIST.



(b) Conv-FC Confusion Matrix for MNIST.

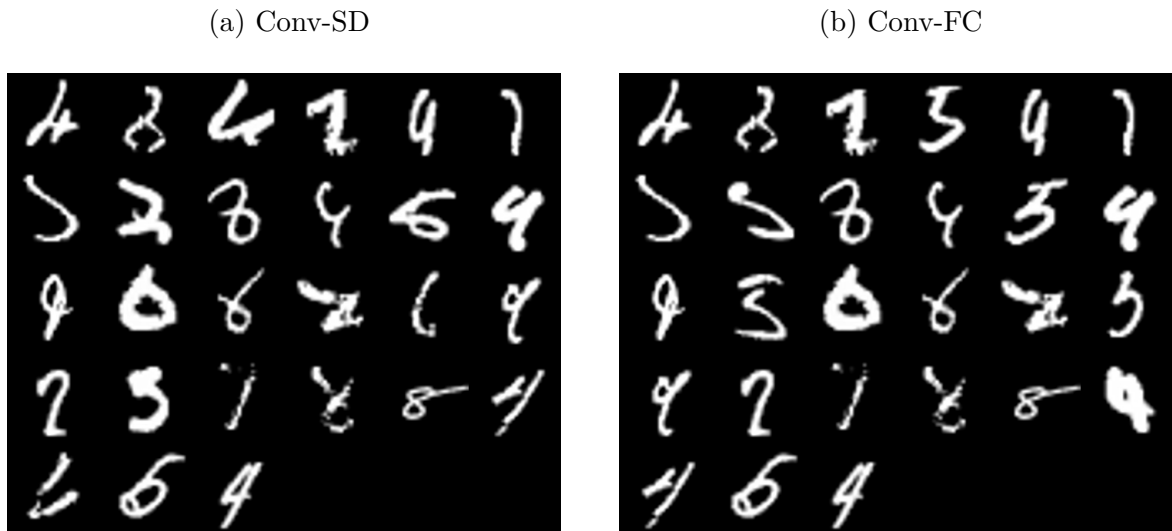


In this plots we consider the model with the best accuracy achieved at the experiments.

25, therefore, even using committees with several Conv-SD to perform the classification,

we will not have certainty that all these digits would be correctly classified. The mean accuracy value reached by a committee of 5 Conv-SD models randomly taken from the 100 trained models was 97.74%. In Table 8 it is shown the comparison between classifiers. This comparison includes the mean of Conv-SD obtained in this experiment and the committee mean result. The barcharts of Figure 32a and 32b illustrate the accuracy behavior of 100 committee models with 5 Conv-SD and Conv-FC, respectively.

Figure 30 – Set of Misclassified Images



Digits misclassified by at least 50% of the classifiers.

Conv-FC: Results on MNIST

The same procedure was executed to analyze Conv-FC models. In Figure 27b is illustrated the accuracy obtained in a bar-chart. Similar values were obtained for this architecture. The minimum and maximum accuracy achieved were 0.9961% and 0.9974%, respectively. The mean accuracy was 0.9966% with a standard deviation of 0.0003%, accuracy of 0.9965% has the highest number of occurrences.

In Figure 31 is illustrated the bar-chart of misclassified with different number of occurrences. The number of images, that were misclassified at least one time by some of the 100 classifiers, is 112. The number of digits misclassified at least by 15% and at most 25% of the classifiers is 6. The misclassified digits with more than 25% and at most 50% are 20. The number of images misclassified by more than 50% of the classifiers was 23. The set of hard digits for Conv-FC is illustrated in Figure 30b.

Table 8 summarizes the results for Conv-FC and Conv-SD models. Additionally, results for committees of these models are presented.

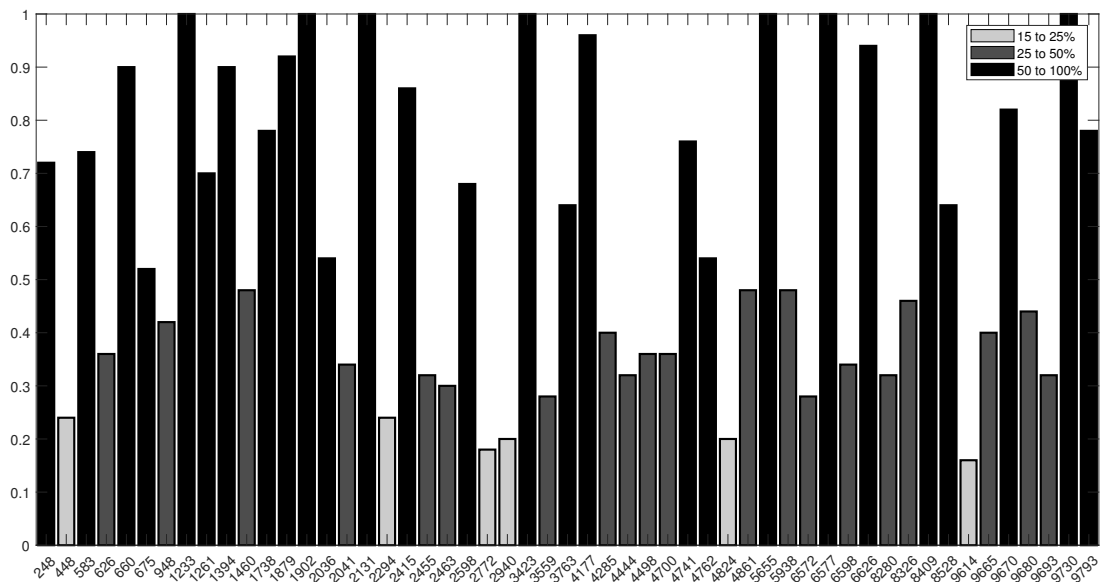
Note that, Conv-FC and Conv-SD have similar values in the MNIST problem. Both can

be considered state of the art of classifiers with respect to this problem. Nevertheless, it is important to test both classifiers in more difficult problems.

6.4 Visual Self-Localization Dataset

The problem of estimating a robot’s location relative to its environment is an issue that can be related with supervised learning, in particular with classification. The strategies to achieve an estimation may vary significantly depending on the type and

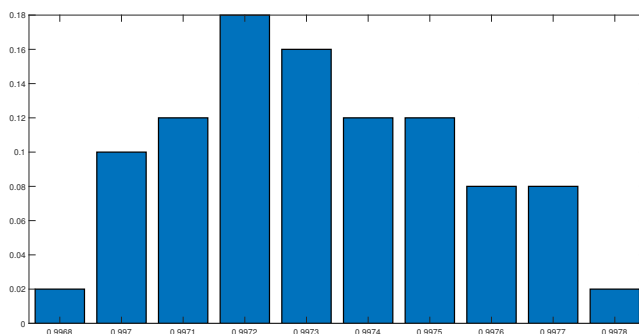
Figure 31 – Conv-FC Bar-chart Common Errors for the Models in the MNIST Experiment



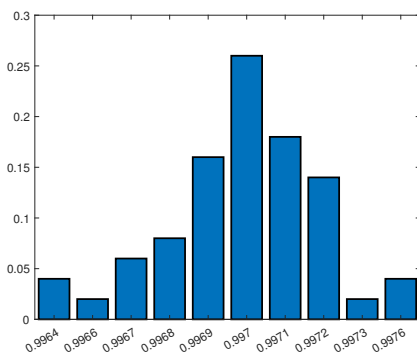
The histogram shows the frequency of the images misclassified.

Figure 32 – Results of Committees for Classification

(a) Committees of 5 Conv-SD



(b) Committees of 5 Conv-FC

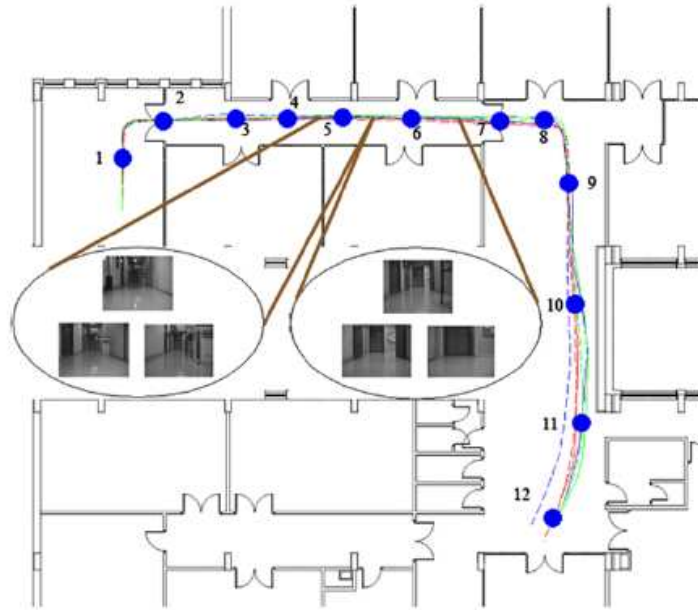


MNIST Accuracy Bar-charts for committees of five classifiers.

Table 8 – MNIST Digits Dataset Results

	Conv-SD	Conv-FC	COMM	Conv-SD	COMM	Conv-FC
Average	99.68	99.66	99.74			99.70

Figure 33 – Design of the VBSL-UPV experiment



Variation on the walks of the selected path inside the laboratory and position of landmarks used to split regions. Source: (GRAÑA et al., 2012)

quantity of features used. We tested the image dataset for vision-based self-localization in mobile robotics provided by the Universidad del Pais Vasco (VBSL-UPV) (COUNTRY., ; VILLAVERDE; GRAÑA; D'ANJOU, 2006). The VBSL-UPV is a dataset for supervised learning with eleven classes and 2265 grayscale images with a resolution of 79×61 .

In this experiment, the location-area of a mobile robot is identified based on a visual system. Typical applications in robotics require a robot to move around inside a building looking for certain areas and locations. The VBSL-UPV dataset was collected to simulate the task of recognizing indoor areas using a single mobile robot with an onboard camera. The process to generate the dataset is given below.

First, a path along the laboratory was determined so that a sequence of images is captured with an onboard optical camera, each time the mobile robot drives this path. Then, some landmark positions were selected to divide the path into eleven regions without gaps. Doors, hallway crossings, printers and windows were considered in order to select these landmarks. Then, each sequence of images was split into eleven subsequences. Each subsequence

Table 9 – Number of Observations in the VBSL-UPV dataset.

	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th	11th	Total
Walk 1	32	15	37	30	29	45	21	49	36	36	32	362
Walk 2	30	9	33	26	27	42	34	58	27	32	39	357
Walk 3	19	27	38	36	35	47	38	59	36	35	34	404
Walk 4	33	22	30	40	30	45	33	49	35	26	42	385
Walk 5	26	22	39	38	36	46	39	41	27	34	35	383
Walk 6	32	21	35	33	32	39	36	50	30	32	34	374
Total	172	116	212	203	189	264	201	306	191	195	216	2265

The number of images by class and by walk collected offline by the mobile robot’s camera.

represents a region. The data were collected traveling the path six times and each travel is referred to as a walk. Therefore, the sequences of images collected using these six walks were all partitioned into eleven subsequences using landmark references. Hence, classes are based on the physical distances to the selected landmark positions. The class of each image is given by the subsequence in which the image is included. This is, divisions are used as a ground truth. Table 9 shows a summary of the performance obtained by walk and some details of the observations contained this experiment.

The source of variation in this experiment are illumination and slight changes in the walks carried out by the robot. Figure 33 illustrates a graphical representation of the experiment and shows the landmarks. Note that slight variations in the path were considered in each walk. The hardest problems of the VBSL-UPV dataset are images corresponding to the boundaries of regions assigned to consecutive landmark positions, because they are naturally very similar. These boundary images are an unavoidable source of confusion for any classifier.

Some classifiers tested for the VBSL-UPV dataset were reported by (ESMI et al., 2014). In order to extend this experiment, we followed the methodology described in (SUSSNER et al., 2012; ESMI et al., 2014). This is, the first and second walk are used in the training and validation stages and walks 3 to 6 make up the test data. Note that, for θ -FAMs the parameters are reference images for each class and the set of these selected images acts as weights of the model. In the training stage, subsets of images from the Walk 1 are validated as hyperparameters in the walk 2. Then, The reference images for the θ -fam is computed selecting the subset of images from the Walk 1 with better accuracy validation in the Walk 2. Since in SGD we do not select reference images, this process is not necessary for SGD related algorithms, therefore, we consider the Walk 1 and Walk 2 as a training data for SGD related models.

In order to perform an extended experiment, we included some classifiers specialized in images such as the AE (HINTON; SALAKHUTDINOV, 2006), HMLP-EL and CNN (see 3.5). Besides, three models proposed here were tested, the SD, FC-SD and a Conv-SD. The CNN and Conv-SD networks were configured with the same architecture except the last

Table 10 – Computed Hyperparameters for VSLR-UPV dataset.

	Parameters		Description
HMLP-EL	LR	800	Introduced in (SUSSNER; CAMPIOTTI, 2020).
	Morph Units	800	
	Linear Units HL2	.5	
	Reg Parameter FC2		
AE (2)	LR	0.001	Two FC Hidden layers. The first hidden layer with 200 nodes and the second one with 50 nodes. Sparsity and L2-regularization for the weights were applied to hidden layers. A CE loss function was considered at training stage. A final fine-tuning (FT) was applied to the whole net.
	Epochs HL1	400	
	Epochs HL2	200	
	Epochs FT	400	
	Reg. FC1	0.004	
	Reg. FC2	0.002	
AE (3)	LR	0.001	Three FC Hidden layers. The first hidden layer with 200 nodes and the second one with 100 nodes and the third one with 50 nodes. Sparsity and L2-regularization were applied to hidden layers. A CE loss function was considered in training stage and a final fine-tuning was applied to the whole net.
	Epochs HL1	400	
	Epochs HL2	200	
	Epochs HL3	200	
	Epochs FT	400	
	Reg. HL1	0.004	
	Reg. HL2	0.002	
Reg. HL3	0.001		
SD	L.R.	0.005	An SD layer with 11 nodes. Each node represents a class. A softmax with a cross entropy function was used in the training stage.
	Epochs	120	
	SGD Method	SGDM	
FC-SD	L.R	0.001	A FC layer with 256 nodes followed by an SD layer. A CE loss function was used in training stage.
	Epochs	200	
	SGD Method	SGDM	
Conv-SD	L.R.	0.001	The convolutional part "Conv" of the network has 15 layers. The classification part has an FC hidden layer with 300 nodes. Then, a BN layer and a ReLu layer. Next, an SD layer with 11 nodes is considered at the classification stage of the CNN. Each node of the SD layer represents a class. A CE loss function was used in training stage.
	Epochs	80	
	Pooling	AV	
	SGD Method	ADAM	
Conv-FC	L.R.	0.001	The convolutional part "Conv" of the network has 15 layers. A FC hidden layer with 300 nodes and a BN with a ReLu activation and an SD layer with 11 nodes were considered at the classification stage of the CNN. Each node of the SD layer represents a class. A CE loss function was used in training stage.
	Epochs	80	
	Pooling	Max	
	SGD Method	ADAM	

layer, in which an FC layer and a morphological SD layer are set for Conv-FC and Conv-SD models respectively. A random search was performed using a k -fold cross validation, with $k = 10$ partitioned randomly, in order to determine the hyperparameters required for the models. In this case, 512 trials were tested and the mean accuracy value of the cross validation is used to select the best set of hyperparameters. The configurations used to set up each model are detailed in Table 10.

The Conv part of the classifiers in which convolutional layers are involved has a similar architecture: First an input layer computes identity function. The second layer is convolution layer with 40 filters of size 3×3 followed by BN and a ReLu activation. Next, a pooling layer of size 3×3 with a stride size of 3 performs downsampling. The sixth layer is convolutional and has 8 filters 3×3 sized followed by a BN and a ReLu activation. The ninth layer is a pooling layer of size 3×3 and stride 3. The remain part of the network consist of a FC with 300 nodes followed by BN, ReLu and dropout layer. The classification task in Conv-FC and Conv-SD is performed by an FC layer and an SD layer respectively. On the one hand, a significant difference found in the validation stage is that, average pooling layers showed best performance in Conv-SD. On the other hand, max pooling layers achieved

Table 11 – Results on the VBSL-UPV dataset.

	Walk-3	Walk-4	Walk-5	Walk-6	Average
Dual S^{\cup} -Fam	0.78	0.72	0.78	0.77	0.76
Dual S^{\cap} -Fam	0.79	0.72	0.79	0.77	0.77
AE-2	0.8292	0.7351	0.8956	0.8342	0.8235
AE-3	0.8936	0.7481	0.9034	0.8877	0.8582
HMLP-EL	0.8960	0.8649	0.9138	0.8850	0.8900
SD	0.9282	0.8649	0.9269	0.8930	0.9033
FC-SD	0.9703	0.8935	0.9504	0.9198	0.9335
Conv-SD	0.9777	0.9610	0.9608	0.9893	0.9722
Conv-FC	0.9431	0.8857	0.9556	0.9198	0.9260

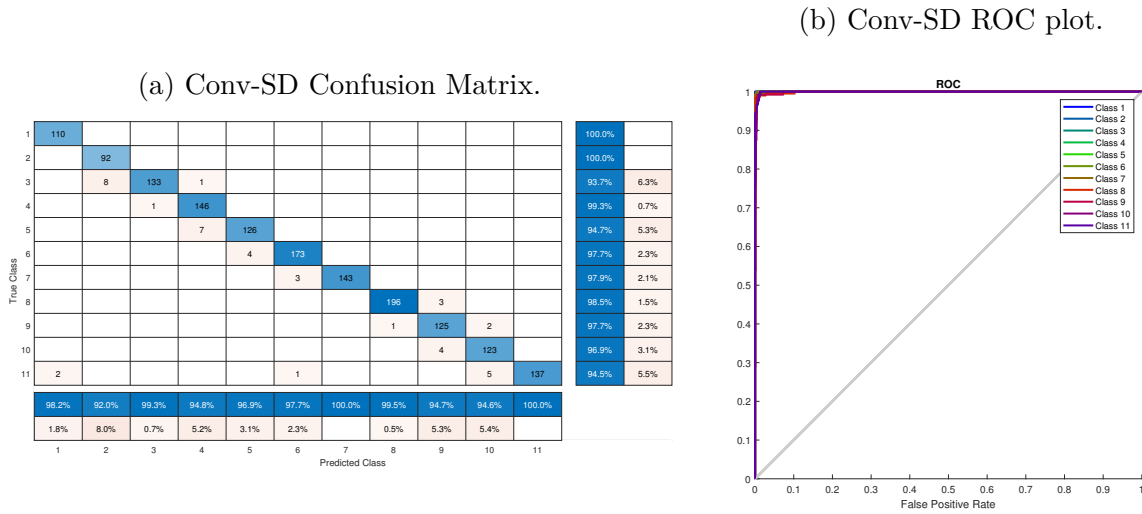
For the AE-2, AE-3, HMLP-EL, SD, FC-SD, Conv-SD and Conv-FC the average accuracy of 5 individual experiments was considered in this comparison.

best performance in the Conv-FC case. For this experiment, an online data-augmentation performs aleatory scaling in the range -0.05% to 0.05% , aleatory X-translation in the range -10 to pixels, aleatory X-shear and Y-shear transformations in the range of -3 to 3 degrees. Besides, contrast and brightness of the original images were modified with a low and a high intensity factor of 1% using an offline strategy, doubling the amount of data for training. This strategy was applied to the SD, FC-SD, Conv-SD and Conv-FC methods.

Table 11 shows the accuracy by walk and the average obtained by 5 analyzed classifiers. The first observation is that all the new models introduced in this experiments outperformed the results presented in (ESMI et al., 2014). In this dataset the best result was obtained by the Conv-SD model. The second best result was obtained by the FC-SD model. Third and fourth place are for the Conv-FC and SD classifiers. Note that, although the Conv-FC model is more complex than FC-SD model, FC-SD outperformed the classical convolutional net in this case. This may have occurred due to the nature of variations in the VBSL-UPV dataset. This is, slight illumination and translation variations were easily captured by the Conv-FC leading to more unstable local minima and overtraining. Nevertheless, the inclusion measure nature of the Conv-SD model captured these translation and illumination variations, which implies that an SD layer played a positive role in this specific problem. Lastly, Figures 34a and 34b illustrate the confusion matrix and the ROC plot for the Conv-SD model with the median accuracy, respectively. The median accuracy of the analyzed model is 97.28% . The confusion chart reveals that most of the misclassified observations lie in consecutive classes, this means that, as expected, most difficult observations are in the boundaries of the regions.

Besides, a ROC curve analysis showed that classes 8 and 11 were harder to classify for this model. Figure 34b illustrates the ROC curve using a threshold of 0.5 for the probability output of the softmax layer. Nevertheless, this result encourages us to test the

Figure 34 – ROBOTICS Analysis of the Results



VBSL-UPV dataset simulations results for Conv-SD.

Conv-SD model in other datasets. Finally, the user may observe that in order to make a fair comparison, in this case we did not use the notion of committees.

6.5 CIFAR-10 Dataset

The CIFAR-10 dataset consists of 60,000, 32×32 color images with ten classes. The labels of the images are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Each image contains exactly one object of these available labels.

The dataset was originally partitioned into training and test sets by Krizhevsky et al. The training and test sets consist of 50,000 and 10,000 images, respectively (KRIZHEVSKY; HINTON et al., 2009). Figure 35 exemplifies some images of the CIFAR-10 dataset.

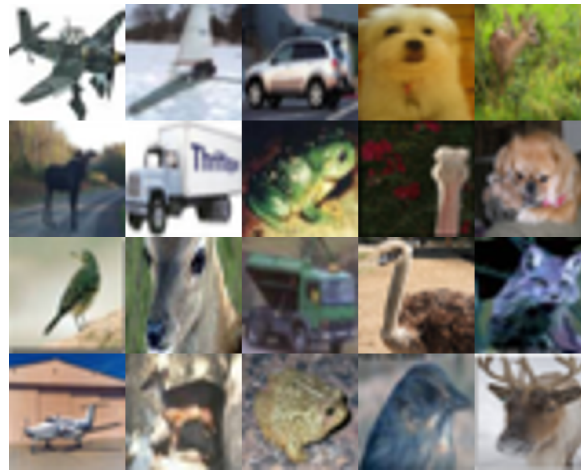
In this experiment we test a Conv-SD model and in the results are considered 5 Conv-SD that were fully trained from scratch. Table 13 shows the general description of the Conv-SD model considered in this experiment. The mean accuracy was 90.77%, considering 5 simulations. The maximum and the minimum accuracy obtained at these simulations were 90.88% and 90.63%, respectively. The performance achieved by these classifiers were: 90.63%, 90.74%, 90.76%, 90.85% and 90.88%.

For the comparative study we used some well known models in DL, such as maxout networks (GOODFELLOW et al., 2013), convolutional networks with dropout regu-

Table 12 – Results on the Dataset CIFAR-10 (Error rate %).

	Conv-SD	Comm	Maxout	DROP	NIN	ML-DNN	ALL-CNN
Error	9.23	8.45	9.38	9.32	8.81	8.12	7.25
Parameters	>6M	>18M	>6M	> 6M	>6M	> 6M	>1M

Figure 35 – Examples of CIFAR-10 Images.



CIFAR-10 observations are 32×32 RGB images of 10 different types of objects.

larization (DROP) (WAN et al., 2013), network in network (NIN) (LIN; CHEN; YAN, 2013), multi-loss regularized deep neural networks (ML-DNN) (XU et al., 2015) and the all-convolutional neural network model (ALL-CNN) (SPRINGENBERG et al., 2014). Besides, in order to perform a fair comparison, we only considered results with data augmentation.

Finally, a committee of classifiers with 3 Conv-SD was included in the comparative study. Table 12 summarizes the error rate obtained in the simulations.

ALL-CNN model outperformed the other classifiers achieving an error rate of 7.25% in the test stage. Multi-loss regularized deep neural network obtained the second best performance. A committee of 3 Conv-SD obtained the third place and individual Conv-SDs achieved an error rate of 9.23%. As expected, the method presented in this work showed a competitive result in this particular dataset. The reader may note that most of the characteristics of ALL-CNN and MLR-DNN models and NIN are compatible with the Conv-SD model and must be studied in future works. Finally, the confusion matrices for the best individual Conv-SD result and the committee of 3 Conv-SD are illustrated in Figure 36a and Figure 36b, respectively.

Table 13 – Configuration of the Conv-SD models for CIFAR-10

1	'imageinput'	Image Input	32x32x3 images with 'zerocenter' normalization
2	'conv1 ₁ '	Convolution	64 3x3x3 convolutions with stride [1 1]
3	'batchnorm ₁ '	Batch Normalization	Batch normalization with 64 channels
4	'ReLU ₁ '	ReLU	ReLU
5	'conv1 ₂ '	Convolution	64 3x3x64 convolutions with stride [1 1]
6	'batchnorm ₂ '	Batch Normalization	Batch normalization with 64 channels
7	'ReLU ₂ '	ReLU	ReLU
8	'maxpool ₁ '	Max Pooling	2x2 max pooling with stride [2 2]
9	"	Dropout	40% dropout
10	'conv2 ₁ '	Convolution	128 3x3x64 convolutions with stride [1 1]
11	'batchnorm ₃ '	Batch Normalization	Batch normalization with 128 channels
12	'ReLU ₃ '	ReLU	ReLU
13	'conv2 ₂ '	Convolution	128 3x3x128 convolutions with stride [1 1]
14	'batchnorm ₄ '	Batch Normalization	Batch normalization with 128 channels
15	'ReLU ₄ '	ReLU	ReLU
16	'maxpool ₂ '	Max Pooling	2x2 max pooling with stride [2 2]
17	"	Dropout	40% dropout
18	'conv3 ₁ '	Convolution	256 3x3x128 convolutions with stride [1 1]
19	'batchnorm ₅ '	Batch Normalization	Batch normalization with 256 channels
20	'ReLU ₅ '	ReLU	ReLU
21	'conv3 ₂ '	Convolution	256 3x3x256 convolutions with stride [1 1]
22	'batchnorm ₆ '	Batch Normalization	Batch normalization with 256 channels
23	'ReLU ₆ '	ReLU	ReLU
24	'maxpool ₃ '	Max Pooling	2x2 max pooling with stride [2 2]
25	'conv3pool'	Convolution	512 2x2x256 convolutions with stride [2 2]
26	'batchnorm ₇ '	Batch Normalization	Batch normalization with 512 channels
27	'ReLU ₇ '	ReLU	ReLU
28	'dropout ₃ '	Dropout	40% dropout
29	'fc ₁ '	Fully Connected	2048 fully connected layer
30	'batchnorm ₈ '	Batch Normalization	Batch normalization with 2048 channels
31	'ReLU ₈ '	ReLU	ReLU
32	'fc ₂ '	Fully Connected	256 fully connected layer
33	'batchnorm ₉ '	Batch Normalization	Batch normalization with 256 channels
34	'ReLU ₉ '	ReLU	ReLU
35	'SD-Layer'	morphologicalClassificationLayerSD	Morphological Classification Layer
36	'softmax'	Softmax	softmax
37	'classoutput'	Classification Output	crossentropyex with 'airplane' and 9 other classes

Conv-SD with 37 layers for 32x32 RGB images.

Figure 36 – CIFAR-10 Confusion Matrices

(a) Best Conv-SD result

airplane	909	7	15	3	1		4	6	22	14	90.9%	7.2%
automobile	2	974						2	5	17	97.4%	2.6%
bird	22		889	12	17	11	31	12	3	3	88.9%	11.1%
cat	14	5	34	761	24	72	52	12	11	15	76.1%	23.9%
deer	3	3	25	9	913	5	26	12	2	2	91.3%	8.7%
dog	9	7	16	64	25	822	24	24	3	6	82.2%	17.8%
frog	4	1	3	6	5		975	2		4	97.5%	2.5%
horse	7	1	3	3	17	13	2	946	2	4	94.6%	5.2%
ship	40	8	4			1	1		937	9	93.7%	6.3%
truck	9	38					1		11	941	94.1%	5.9%
	89.4%	93.3%	89.9%	86.7%	91.1%	89.0%	87.4%	93.1%	94.1%	92.7%		
	10.6%	6.7%	10.1%	11.3%	8.9%	11.0%	12.6%	6.9%	5.9%	7.3%		
	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck		
	Predicted Class											

(b) Committee of three Conv-SDs

1	936	8	13	3	1		5	4	19	11	93.6%	6.4%
2	3	974					1	1	3	18	97.4%	2.6%
3	15		888	12	18	12	27	10	4	4	88.8%	10.2%
4	13	5	28	774	27	62	51	12	10	18	77.4%	22.6%
5	3	3	17	7	923	5	28	11	2	1	92.3%	7.7%
6	9	5	15	66	26	826	24	20	2	7	82.6%	17.4%
7	5		2	3	5		979	2		4	97.9%	2.1%
8	7		2	2	14	11	2	957	2	3	95.7%	4.3%
9	32	6	2			1	2		945	12	94.5%	5.5%
10	10	36					1		10	943	94.3%	5.7%
	90.6%	93.9%	91.9%	89.3%	91.0%	90.1%	87.4%	94.1%	94.8%	92.4%		
	9.4%	6.1%	8.1%	10.7%	9.0%	9.9%	12.6%	5.9%	5.2%	7.6%		
	1	2	3	4	5	6	7	8	9	10		
	Predicted Class											

7 Conclusions

In this work, we extended the set of operations that are used in MPs to the domain of ℓ -rings. The study of this set of operations in ℓ -rings and the analysis of the behavior of MP models led us to the definition of two operators: \mathbb{I} -set descriptors ϕ and aggregators \bigoplus . Then, we described the foundations of the GMP model and we performed a mathematical review of GMP units of computations based on the aforementioned pairs of operators. As a result, a feedforward network that makes use of P-descriptor pairs (\bigoplus, ϕ) as units of computation was fully developed. Furthermore, it was demonstrated that units of computation of MPs and DMNNs fit the definition of GMP whenever P-descriptor pairs $P = (\bigwedge, \phi_{\wedge})$ are considered. Therefore, from this point of view GMPs are more general than traditional MPs. Other results include a link between some P-descriptors and some traditional distances, whenever an archimedean o-ring is considered.

Moreover, using the P-descriptor pair $P = (\sum, \phi_{\times})$, we described a unit of computation in which differentiation is possible. This allowed us to develop the concept of GMP layers in terms of the DL framework. Specifically, in feedforward DL models for supervised classification trained by SGD. As mentioned before, the generalization proposed here considers the units of computation of MP and DMNN models extending the set of possible operations in the MP framework. Nevertheless, a stronger relationship between the multiplication operation in ℓ -rings and mathematical morphology must be studied in future research.

In this work, we described two layers. The first one is a fully connected layer with units that compute P-descriptor mappings and the other one performs an aggregation by class. However, we focused on the first type of layer, setting one unit of computation for each class. Multiple unit of computation and class aggregation strategies are matters that are subject to further research.

The next step was to define some models that make use of the aforementioned layers. Therefore, we incrementally built three abstract architectures: the SD, the FC-SD and the Conv-SD. The resulting models were experimentally tested in some simulations. The first set of simulations was taken from the UCI Machine Learning Repository. The achieved results reveal the competitive performance of the SD and FC-SD models. An additional simulation was performed on the MNIST1000 dataset. The results reveal a competitive performance for all the classifiers considered in this simulation. In particular, the Conv-SD classifier outperformed all the other classifiers. A more extensive research on a hybridization of HLMP-EL and the models presented in this thesis must be considered for future research efforts.

Simulations on the MNIST digits and the VRSL-UPV datasets revealed an excellent

generalization capability of our models, in particular the Conv-SD model. These results encourage us to conduct experiments using other datasets and to continue this research in future works.

As a final consideration, we remark that DL is a growing trend in machine learning. Therefore, new algorithms, layers, models and techniques to improve the quality of DL classifiers are reported every month. The user may note that we did not consider some developments of DL framework. For instance, Maxout ([GOODFELLOW et al., 2013](#)), NIN ([LIN; CHEN; YAN, 2013](#)) or multi-loss regularization ([XU et al., 2015](#)) are some examples of models, in which the use of SD layers is possible. In future work, it is extremely important to explore how the results presented in this thesis fit these aforementioned developments in DL.

Bibliography

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANÉ, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCHE, V.; VASUDEVAN, V.; VIÉGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Cited on page 70.

AEBERHARD, S.; COOMANS, D.; VEL, O. D. Comparative analysis of statistical pattern recognition methods in high dimensional settings. *Pattern Recognition*, Elsevier, v. 27, n. 8, p. 1065–1077, 1994. Cited on page 123.

ALTMAN, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, Taylor & Francis Group, v. 46, n. 3, p. 175–185, 1992. Cited on page 121.

ANDERSON, M.; FEIL, T. *Lattice-ordered groups: an introduction*. [S.l.]: Springer Science & Business Media, 2012. v. 4. Cited 4 times on pages 29, 33, 34, and 35.

ARAÚJO, R. d. A. A class of hybrid morphological perceptrons with application in time series forecasting. *Knowledge-Based Systems*, Elsevier, v. 24, n. 4, p. 513–529, 2011. Cited on page 105.

ARAÚJO, R. d. A.; OLIVEIRA, A. L.; MEIRA, S. R. A dilation-erosion-linear perceptron for bovespa index prediction. In: SPRINGER. *International Conference on Intelligent Data Engineering and Automated Learning*. [S.l.], 2012. p. 407–415. Cited 4 times on pages 16, 20, 104, and 121.

ARCE, F.; ZAMORA, E.; FÓCIL-ARIAS, C.; SOSSA, H. Dendrite ellipsoidal neurons based on k-means optimization. *Evolving Systems*, Springer, v. 10, n. 3, p. 381–396, 2019. Cited on page 102.

ARCE, F.; ZAMORA, E.; HERNÁNDEZ, G.; SOSSA, H. Efficient lane detection based on artificial neural networks. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, Copernicus GmbH, v. 4, p. 13, 2017. Cited 2 times on pages 19 and 99.

ARCE, F.; ZAMORA, E.; SOSSA, H.; BARRÓN, R. Differential evolution training algorithm for dendrite morphological neural networks. *Applied Soft Computing*, v. 68, p. 303 – 313, 2018. ISSN 1568-4946. Cited 2 times on pages 99 and 103.

ARKHANGEL'SKII, A. V. *General topology I*. Berlin New York: Springer-Verlag, 1990. ISBN 3540181784. Cited on page 37.

ARPIT, D.; ZHOU, Y.; NGO, H.; GOVINDARAJU, V. Why regularized auto-encoders learn sparse representation? *arXiv preprint arXiv:1505.05561*, 2015. Cited on page 81.

ARTHUR, D.; VASSILVITSKII, S. *K-means++: The advantages of careful seeding*. [S.l.], 2006. Cited on page 99.

ASCH, V. V. Macro-and micro-averaged evaluation measures. *Belgium: CLiPS*, v. 49, 2013. Cited on page 126.

ASUNCION, A.; NEWMAN, D. *UCI Machine Learning Repository*. 2007. Cited 3 times on pages 120, 122, and 123.

AUDA, G.; KAMEL, M. Modular neural networks: a survey. *International Journal of Neural Systems*, World Scientific, v. 9, n. 02, p. 129–151, 1999. Cited on page 18.

BAGIROV, A.; RUBINOV, A.; SOUKHOROUKOVA, N.; YEARWOOD, J. Unsupervised and supervised data classification via nonsmooth and global optimization. *Top*, Springer, v. 11, n. 1, p. 1–75, 2003. Cited on page 125.

BANON, G. J. F.; BARRERA, J. Decomposition of mappings between complete lattices by mathematical morphology, part I. general lattices. *Signal Processing*, Elsevier, v. 30, n. 3, p. 299–327, 1993. Cited 4 times on pages 17, 40, 91, and 95.

BERNAU, S. J. Unique representation of archimedean lattice groups and normal archimedean lattice rings. *Proceedings of the London Mathematical Society*, Oxford University Press, v. 3, n. 1, p. 384–384, 1966. Cited on page 35.

BIRKHOFF, G. *Lattice theory*. [S.l.]: American Mathematical Soc., 1940. v. 25. Cited on page 23.

_____. *Lattice theory*. [S.l.]: American Mathematical Society, 1967. Cited 2 times on pages 23 and 29.

_____. Lattice-ordered groups. *Selected Papers on Algebra and Topology by Garrett Birkhoff*, Springer Science & Business Media, v. 43, n. 2, p. 412, 1987. Cited 2 times on pages 20 and 34.

BOOLE, G.; GRATTAN-GUINNESS, I.; BORNET, G. George Boole: selected manuscripts on logic and its philosophy. 1999. Cited on page 22.

BORKOTOKEY, S.; KOMORNÍKOVÁ, M.; LI, J.; MESIAR, R. Aggregation functions, similarity and fuzzy measures. In: SPRINGER. *International Summer School on Aggregation Operators*. [S.l.], 2017. p. 223–228. Cited on page 44.

BRYSON, A. E. A gradient method for optimizing multi-stage allocation processes. In: *Proc. Harvard Univ. Symposium on digital computers and their applications*. [S.l.: s.n.], 1961. v. 72. Cited on page 53.

BÜHLMANN, P.; GEER, S. V. D. *Statistics for high-dimensional data: methods, theory and applications*. [S.l.]: Springer Science & Business Media, 2011. Cited 2 times on pages 69 and 126.

CALVO, T.; MAYOR, G.; MESIAR, R. *Aggregation operators: new trends and applications*. [S.l.]: Physica, 2012. v. 97. Cited on page 44.

CESTNIK, B.; KONONENKO, I.; BRATKO, I. Assistant 86: A knowledge-elicitation tool for sophisticated users. In: *Proceedings of the 2nd European Conference on European Working Session on Learning*. [S.l.: s.n.], 1987. p. 31–45. Cited on page 125.

- CHARISOPOULOS, V.; MARAGOS, P. Morphological perceptrons: geometry and training algorithms. In: SPRINGER. *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*. [S.l.], 2017. p. 3–15. Cited 2 times on pages 106 and 107.
- CHEN, T.; GUESTRIN, C. Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. [S.l.: s.n.], 2016. p. 785–794. Cited on page 121.
- CHEN, T.; HE, T.; BENESTY, M.; KHOTILOVICH, V.; TANG, Y. Xgboost: extreme gradient boosting. *R package version 0.4-2*, p. 1–4, 2015. Cited on page 121.
- CHOLLET, F. et al. *Keras*. [S.l.]: GitHub, 2015. <<https://github.com/fchollet/keras>>. Cited on page 70.
- CORTES, C.; VAPNIK, V. Support-vector networks. *Machine learning*, Springer, v. 20, n. 3, p. 273–297, 1995. Cited on page 121.
- COUNTRY., C. I. G. of the University of the B. *Image Database for Vision-Based Self-Localization in Mobile Robotics*. <<http://www.ehu.es/ccwintco/index.php/Pioneer>>, year=. Cited 2 times on pages 120 and 135.
- CUNINGHAME-GREEN, R. Minimax algebra and applications. *Fuzzy Sets and Systems*, Elsevier, v. 41, n. 3, p. 251–267, 1991. Cited on page 88.
- CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, Springer, v. 2, n. 4, p. 303–314, 1989. Cited 2 times on pages 53 and 76.
- DAN, Y.; HU, B. Q.; QIAO, J. General L-fuzzy aggregation functions based on complete residuated lattices. *Soft Computing*, Springer, p. 1–26, 2020. Cited 2 times on pages 39 and 44.
- DARNEL, M. *Theory of lattice-ordered groups*. [S.l.]: CRC Press, 1994. v. 187. Cited on page 34.
- DAVIDSON, J. L.; HUMMER, F. Morphology neural networks: An introduction with applications. *Circuits, Systems and Signal Processing*, Springer, v. 12, n. 2, p. 177–210, 1993. Cited on page 88.
- DAVIDSON, J. L.; SUN, K. Template learning in morphological neural nets. In: *image algebra and morphological image processing II*. [S.l.]: International Society for Optics and Photonics, 1991. v. 1568, p. 176–187. Cited 2 times on pages 17 and 39.
- DEMIR, C.; YENER, B. Automated cancer diagnosis based on histopathological images: a systematic survey. *Rensselaer Polytechnic Institute, Tech. Rep*, Citeseer, 2005. Cited on page 123.
- ESMI, E. L.; SUSSNER, P.; BUSTINCE, H.; FERNÁNDEZ, J. Theta-fuzzy associative memories (Theta-FAMs). *IEEE Transactions on Fuzzy Systems*, IEEE, v. 23, n. 2, p. 313–326, 2014. Cited 4 times on pages 16, 87, 136, and 138.

- ESPOSITO, F.; MALERBA, D.; SEMERARO, G. Multistrategy learning for document recognition. *Applied Artificial Intelligence an International Journal*, Taylor & Francis, v. 8, n. 1, p. 33–84, 1994. Cited on page 123.
- FISHER, R. A. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, Wiley Online Library, v. 7, n. 2, p. 179–188, 1936. Cited on page 121.
- FLEISCHER, I. The Hahn embedding theorem: analysis, refinements, proof. In: *Algebra Carbondale 1980*. [S.l.]: Springer, 1981. p. 278–290. Cited on page 35.
- FORINA, M.; LEARDI, R.; ARMANINO, C.; LANTERI, S.; CONTI, P.; PRINCI, P. Parvus: An extendable package of programs for data exploration, classification and correlation. *Journal of Chemometrics*, v. 4, n. 2, p. 191–193, 1988. Cited on page 123.
- FREY, P. W.; SLATE, D. J. Letter recognition using holland-style adaptive classifiers. *Machine learning*, Springer, v. 6, n. 2, p. 161–182, 1991. Cited on page 123.
- FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. *The elements of statistical learning*. [S.l.]: Springer series in statistics New York, 2001. v. 1. Cited on page 126.
- FUKUSHIMA, K.; MIYAKE, S. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In: *Competition and cooperation in neural nets*. [S.l.]: Springer, 1982. p. 267–285. Cited on page 54.
- GADER, P. D.; KELLER, J. M.; NELSON, B. N. Recognition technology for the detection of buried land mines. *IEEE Transactions on Fuzzy Systems*, IEEE, v. 9, n. 1, p. 31–43, 2001. Cited on page 16.
- GANTER, B.; STUMME, G.; WILLE, R. *Formal concept analysis: foundations and applications*. [S.l.]: springer, 2005. v. 3626. Cited on page 39.
- GANTER, B.; WILLE, R. *Formal concept analysis: mathematical foundations*. [S.l.]: Springer Science & Business Media, 2012. Cited on page 39.
- GLASS, A. M. W. *Partially ordered groups*. [S.l.]: World Scientific, 1999. v. 7. Cited on page 34.
- GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2010. p. 249–256. Cited 5 times on pages 69, 71, 73, 75, and 85.
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep learning*. [S.l.]: MIT press, 2016. Cited 7 times on pages 19, 55, 66, 67, 75, 80, and 85.
- GOODFELLOW, I. J.; WARDE-FARLEY, D.; MIRZA, M.; COURVILLE, A.; BENGIO, Y. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013. Cited 2 times on pages 139 and 144.
- GOUTSIAS, J.; HEIJMANS, H. J. *Mathematical morphology*. [S.l.]: IOS press, 2000. Cited on page 40.
- GRAÑA, M. A brief review of lattice computing. In: IEEE. *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. [S.l.], 2008. p. 1777–1781. Cited on page 39.

- GRANA, M. Lattice computing: lattice theory based computational intelligence. In: *Proc. Kosen Workshop on Mathematics, Technology, and Education (MTE)*. [S.l.: s.n.], 2008. p. 19–27. Cited on page 39.
- GRAÑA, M.; GALLEGO, J.; TORREALDEA, F. J.; D'ANJOU, A. On the application of associative morphological memories to hyperspectral image analysis. In: SPRINGER. *International Work-Conference on Artificial Neural Networks*. [S.l.], 2003. p. 567–574. Cited on page 16.
- GRAÑA, M.; VILLAVARDE, I.; LOPEZ-GUEDE, J. M.; FERNANDEZ-GAUNA, B. Lattice independent component analysis for appearance-based mobile robot localization. *Neural Computing and Applications*, Springer, v. 21, n. 5, p. 1031–1042, 2012. Cited on page 135.
- GRÄTZER, G. *General lattice theory*. [S.l.]: Springer Science & Business Media, 2002. Cited on page 23.
- GUVENIR, H. A.; ACAR, B.; DEMIROZ, G.; CEKIN, A. A supervised machine learning algorithm for arrhythmia analysis. In: *Computers in Cardiology*. [S.l.: s.n.], 1997. p. 433–436. Cited on page 124.
- GÜVENIR, H. A.; DEMİRÖZ, G.; ILTER, N. Learning differential diagnosis of erythematous-squamous diseases using voting feature intervals. *Artificial intelligence in medicine*, Elsevier, v. 13, n. 3, p. 147–165, 1998. Cited on page 124.
- HARALICK, R. M.; STERNBERG, S. R.; ZHUANG, X. Image analysis using mathematical morphology. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, n. 4, p. 532–550, 1987. Cited on page 40.
- HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of statistical learning: data mining, inference, and prediction*. [S.l.]: Springer Science & Business Media, 2009. Cited on page 121.
- HE, K.; ZHANG, X.; REN, S.; SUN, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the IEEE international conference on computer vision*. [S.l.: s.n.], 2015. p. 1026–1034. Cited 2 times on pages 73 and 85.
- _____. Deep residual learning for image recognition. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [S.l.: s.n.], 2016. p. 770–778. Cited 2 times on pages 71 and 85.
- HEBB, D. O. *The organization of behavior: A neuropsychological theory*. [S.l.]: Psychology Press, 2005. Cited on page 52.
- HEIJMANS, H. J. *Morphological image operators*. [S.l.]: Academic Press Boston, 1994. v. 4. Cited 2 times on pages 17 and 39.
- HERNÁNDEZ, G.; ZAMORA, E.; SOSSA, H. Morphological-linear neural network. In: IEEE. *2018 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2018. p. 1–6. Cited on page 20.

HINTON, G.; SRIVASTAVA, N.; SWERSKY, K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, v. 14, n. 8, 2012. Cited 2 times on pages 65 and 121.

HINTON, G. E.; SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science*, American Association for the Advancement of Science, v. 313, n. 5786, p. 504–507, 2006. Cited 2 times on pages 81 and 136.

HOCHREITER, S.; BENGIO, Y.; FRASCONI, P.; SCHMIDHUBER, J. et al. *Gradient flow in recurrent nets: the difficulty of learning long-term dependencies*. [S.l.]: A Field Guide to Dynamical Recurrent Neural Networks. IEEE Press, 2001. Cited on page 69.

HONG, Z.-Q.; YANG, J.-Y. Optimal discriminant plane for a small number of samples and design method of classifier on the plane. *Pattern Recognition*, Elsevier, v. 24, n. 4, p. 317–324, 1991. Cited on page 125.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, National Academy Sciences, v. 79, n. 8, p. 2554–2558, 1982. Cited on page 53.

HUANG, G.-B. An insight into extreme learning machines: random neurons, random features and kernels. *Cognitive Computation*, Springer, v. 6, n. 3, p. 376–390, 2014. Cited 2 times on pages 107 and 121.

_____. What are extreme learning machines? filling the gap between Frank Rosenblatt's dream and john von neumann's puzzle. *Cognitive Computation*, Springer, v. 7, n. 3, p. 263–278, 2015. Cited 2 times on pages 107 and 108.

HUANG, G.-B.; ZHU, Q.-Y.; SIEW, C.-K. Extreme learning machine: theory and applications. *Neurocomputing*, Elsevier, v. 70, n. 1-3, p. 489–501, 2006. Cited 2 times on pages 107 and 121.

HUTTER, M.; ZAFFALON, M. Distribution of mutual information from complete and incomplete data. *Computational Statistics & Data Analysis*, Elsevier, v. 48, n. 3, p. 633–657, 2005. Cited on page 125.

IOFFE, S.; SZEGEDY, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. Cited on page 78.

IVAKHNENKO, A. *Cybernetic predicting devices*. [S.l.]. Cited on page 54.

_____. Cybernetics and forecasting techniques. Cited on page 54.

KABURLASOS, V. G.; KEHAGIAS, A. Fuzzy inference system (fis) extensions based on the lattice theory. *IEEE Transactions on Fuzzy Systems*, IEEE, v. 22, n. 3, p. 531–546, 2013. Cited on page 88.

KABURLASOS, V. G.; PETRIDIS, V. Fuzzy lattice neurocomputing (fn) models. *Neural Networks*, Elsevier, v. 13, n. 10, p. 1145–1170, 2000. Cited 2 times on pages 42 and 88.

KALMAN, J. Triangle inequality in l-groups. *Proceedings of the American Mathematical Society*, v. 11, n. 3, p. 395, 1960. Cited on page 34.

- KELLEY, H. J. Gradient theory of optimal flight paths. *Ars Journal*, v. 30, n. 10, p. 947–954, 1960. Cited on page 53.
- KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. Cited on page 65.
- KLEIN, J.; SERRA, J. The texture analyser. *Journal of Microscopy*, Wiley Online Library, v. 95, n. 2, p. 349–356, 1972. Cited on page 16.
- KOCH, C.; SEGEV, I. et al. *Methods in neuronal modeling: from ions to networks*. [S.l.]: MIT press, 1998. Cited on page 93.
- KOŁODYŃSKI, S. IsarMathLib—a Formalized Mathematics Library for Isabelle/zf. In: *CICM Workshops*. [S.l.: s.n.], 2018. Cited on page 36.
- KOMORNÍKOVÁ, M.; MESIAR, R. Aggregation functions on bounded partially ordered sets and their classification. *Fuzzy Sets and Systems*, Elsevier, v. 175, n. 1, p. 48–56, 2011. Cited on page 44.
- KOWSARI, K.; HEIDARYSAFA, M.; BROWN, D. E.; MEIMANDI, K. J.; BARNES, L. E. RMDL: Random multimodel deep learning for classification. In: *Proceedings of the 2nd International Conference on Information System and Data Mining*. [S.l.: s.n.], 2018. p. 19–28. Cited on page 128.
- KRIZHEVSKY, A.; HINTON, G. et al. Learning multiple layers of features from tiny images. Citeseer, 2009. Cited 3 times on pages 71, 120, and 139.
- LECUN, Y.; BENGIO, Y. et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, v. 3361, n. 10, p. 1995, 1995. Cited 4 times on pages 19, 54, 55, and 72.
- LECUN, Y.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKEL, L. D. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989. Cited on page 54.
- LECUN, Y.; BOSER, B. E.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W. E.; JACKEL, L. D. Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1990. p. 396–404. Cited 2 times on pages 73 and 128.
- LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Cited 2 times on pages 54 and 120.
- LECUN, Y.; CORTES, C. MNIST handwritten digit database. 2010. Cited 2 times on pages 120 and 128.
- LECUN, Y.; TOURESKY, D.; HINTON, G.; SEJNOWSKI, T. A theoretical framework for back-propagation. In: CMU, PITTSBURGH, PA: MORGAN KAUFMANN. *Proceedings of the 1988 connectionist models summer school*. [S.l.], 1988. v. 1, p. 21–28. Cited on page 67.

- LIN, M.; CHEN, Q.; YAN, S. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. Cited 2 times on pages 140 and 144.
- LOH, W.-Y. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, Wiley Online Library, v. 1, n. 1, p. 14–23, 2011. Cited on page 127.
- LOH, W.-Y.; SHIH, Y.-S. Split selection methods for classification trees. *Statistica sinica*, JSTOR, p. 815–840, 1997. Cited on page 127.
- MAKHZANI, A.; FREY, B. K-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013. Cited 2 times on pages 66 and 81.
- MALERBA, D.; ESPOSITO, F.; SEMERARO, G. A further comparison of simplification methods for decision-tree induction. In: *Learning from data*. [S.l.]: Springer, 1996. p. 365–374. Cited on page 123.
- MATHERON, G. Random sets and integral geometry. 1975. Cited on page 16.
- MATHERON, G.; SERRA, J. The birth of mathematical morphology. In: SYDNEY, AUSTRALIA. *Proc. 6th Intl. Symp. Mathematical Morphology*. [S.l.], 2002. p. 1–16. Cited on page 16.
- MATLAB. *9.7.0.1190202 (R2019b)*. Natick, Massachusetts: The MathWorks Inc., 2018. Cited 2 times on pages 70 and 71.
- MEHRTENS, H. *Die Entstehung der Verbandstheorie*. [S.l.]: Gerstenberg, 1979. Cited on page 23.
- MEL, B. W. Synaptic integration in an excitable dendritic tree. *Journal of Neurophysiology*, American Physiological Society Bethesda, MD, v. 70, n. 3, p. 1086–1101, 1993. Cited on page 93.
- _____. Why have dendrites? a computational perspective. *Dendrites*, Oxford University Press New York, p. 271–289, 1999. Cited on page 93.
- MINSKY, M.; PAPER, S. *Perceptrons*. [S.l.]: MIT Press, 1969. Cited on page 53.
- MITTAL, S. A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, Springer, p. 1–31, 2018. Cited on page 74.
- MOHAMED, A.-r.; DAHL, G. E.; HINTON, G. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, IEEE, v. 20, n. 1, p. 14–22, 2011. Cited on page 55.
- MURPHY, K. P. *Machine learning: a probabilistic perspective*. [S.l.]: MIT press, 2012. Cited on page 61.
- NAGI, J.; DUCATELLE, F.; CARO, G. A. D.; CIREŞAN, D.; MEIER, U.; GIUSTI, A.; NAGI, F.; SCHMIDHUBER, J.; GAMBARDELLA, L. M. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In: IEEE. *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*. [S.l.], 2011. p. 342–347. Cited on page 74.

- PEDRYCZ, W.; PARK, B.-J.; OH, S.-K. The design of granular classifiers: A study in the synergy of interval calculus and fuzzy sets in pattern recognition. *Pattern Recognition*, Elsevier, v. 41, n. 12, p. 3720–3735, 2008. Cited 3 times on pages 30, 42, and 110.
- PEIRCE, C. S. On the algebra of logic. *American Journal of Mathematics*, JSTOR, v. 3, n. 1, p. 15–57, 1880. Cited on page 22.
- PEREZ, L.; WANG, J. The effectiveness of data augmentation in image classification using deep learning. *ArXiv*, abs/1712.04621, 2017. Cited on page 128.
- PESSOA, L. F.; MARAGOS, P. Morphological/rank neural networks and their adaptive optimal design for image processing. In: IEEE. *1996 IEEE International Conference on Acoustics, Speech, and Signal Processing Conference Proceedings*. [S.l.], 1996. v. 6, p. 3398–3401. Cited 3 times on pages 20, 103, and 105.
- _____. Neural networks with hybrid morphological/rank/linear nodes and their application to handwritten character recognition. In: IEEE. *9th European Signal Processing Conference (EUSIPCO 1998)*. [S.l.], 1998. p. 1–4. Cited on page 18.
- _____. Neural networks with hybrid morphological/rank/linear nodes: a unifying framework with applications to handwritten character recognition. *Pattern Recognition*, v. 33, n. 6, p. 945 – 960, 2000. ISSN 0031-3203. Cited 3 times on pages 18, 103, and 104.
- PESSOA, L. F. C. Nonlinear systems and neural networks with hybrid morphological/rank/linear nodes: Optimal design and applications to image processing and pattern recognition. 1999. Cited 2 times on pages 18 and 20.
- PETRIDIS, V.; KABURLASOS, V. G. Fuzzy lattice neural network (FLNN): a hybrid model for learning. *IEEE Transactions on Neural Networks*, IEEE, v. 9, n. 5, p. 877–890, 1998. Cited on page 88.
- PRASAD, A. M.; IVERSON, L. R.; LIAW, A. Newer classification and regression tree techniques: bagging and random forests for ecological prediction. *Ecosystems*, Springer, v. 9, n. 2, p. 181–199, 2006. Cited on page 121.
- RITTER, G.; DAVIDSON, J.; WILSON, J. Beyond mathematical morphology. In: *Visual Communications and Image Processing II*. [S.l.]: International Society for Optics and Photonics, 1987. v. 845, p. 260–269. Cited on page 17.
- RITTER, G.; IANCU, L. Single layer feedforward neural network based on lattice algebra. In: IEEE. *Proceedings of the International Joint Conference on Neural Networks, 2003*. [S.l.], 2003. v. 4, p. 2887–2892. Cited 2 times on pages 18 and 93.
- RITTER, G. X.; IANCU, L.; URCID, G. Morphological perceptrons with dendritic structure. In: IEEE. *The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ'03*. [S.l.], 2003. v. 2, p. 1296–1301. Cited 5 times on pages 18, 88, 92, 94, and 96.
- RITTER, G. X.; SUSSNER, P. An introduction to morphological neural networks. In: IEEE. *Proceedings of 13th International Conference on Pattern Recognition*. [S.l.], 1996. v. 4, p. 709–717. Cited 5 times on pages 17, 39, 88, 89, and 93.
- RITTER, G. X.; SUSSNER, P.; LEON, J. Diaz-de. Morphological associative memories. *IEEE Transactions on neural networks*, IEEE, v. 9, n. 2, p. 281–293, 1998. Cited on page 16.

RITTER, G. X.; URCID, G. Lattice algebra approach to single-neuron computation. *IEEE Transactions on Neural Networks*, IEEE, v. 14, n. 2, p. 282–295, 2003. Cited on page 18.

_____. Learning in lattice neural networks that employ dendritic computing. In: *Computational Intelligence Based on Lattice Theory*. [S.l.]: Springer, 2007. p. 25–44. Cited 3 times on pages 18, 96, and 97.

RITTER, G. X.; URCID, G.; VALDIVIEZO-N, J.-C. Two lattice metrics dendritic computing for pattern recognition. In: IEEE. *2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*. [S.l.], 2014. p. 45–52. Cited on page 39.

ROBBINS, H.; MONRO, S. A stochastic approximation method. *The annals of mathematical statistics*, JSTOR, p. 400–407, 1951. Cited on page 61.

ROMANUKE, V. V. Training data expansion and boosting of convolutional neural networks for reducing the MNIST dataset error rate. kpi, 2016. Cited 2 times on pages 128 and 129.

RONSE, C. Why mathematical morphology needs complete lattices. *Signal processing*, Elsevier, v. 21, n. 2, p. 129–154, 1990. Cited 2 times on pages 16 and 40.

ROSASCO, L.; VITO, E. D.; CAPONNETTO, A.; PIANA, M.; VERRI, A. Are loss functions all the same? *Neural Computation*, MIT Press, v. 16, n. 5, p. 1063–1076, 2004. Cited on page 61.

ROSENBLATT, F. *Principles of neurodynamics. perceptrons and the theory of brain mechanisms*. [S.l.], 1961. Cited 2 times on pages 19 and 52.

ROTA, G.-C. The many lives of lattice theory. *Notices of the AMS*, v. 44, n. 11, p. 1440–1445, 1997. Cited on page 23.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. *Learning internal representations by error propagation*. [S.l.], 1985. Cited 3 times on pages 53, 54, and 64.

_____. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986. Cited on page 53.

RUSSELL, S. J.; NORVIG, P. *Artificial intelligence: a modern approach*. [S.l.]: Malaysia; Pearson Education Limited,, 2016. Cited on page 55.

SCHMALZ, M. S.; RITTER, G. X. Hyperspectral endmember extraction and signal classification with morphological networks. In: CITESEER. *in Proceedings of the AMOS 2006 Conference, Maui HI*. [S.l.], 2006. Cited on page 16.

SCHMIDHUBER, J. Deep learning in neural networks: An overview. *Neural Networks*, Elsevier, v. 61, p. 85–117, 2015. Cited 3 times on pages 54, 66, and 85.

SCHRÖDER, E. *Vorlesungen über die Algebra der Logik (exakte Logik), von Dr. Ernst Schröder,...* [S.l.]: BG Teubner., 1890. Cited on page 23.

SEGEV, I.; RALL, W. Computational study of an excitable dendritic spine. *Journal of Neurophysiology*, American Physiological Society Bethesda, MD, v. 60, n. 2, p. 499–523, 1988. Cited on page 93.

- SERRA, J. *Image analysis and mathematical morphology*. [S.l.]: Academic Press, Inc., 1983. Cited 2 times on pages 16 and 39.
- _____. *Image Analysis and Mathematical Morphology: Vol.: 2: Theoretical Advances*. [S.l.]: Academic Press, 1988. Cited on page 40.
- SERRA, J.; VERCHERY, G. Mathematical morphology applied to fibre composite materials. *Fibre Science and Technology*, Elsevier, v. 6, n. 2, p. 141–158, 1973. Cited on page 16.
- SHORTEN, C.; KHOSHGOFTAAR, T. M. A survey on image data augmentation for deep learning. *Journal of Big Data*, Springer, v. 6, n. 1, p. 60, 2019. Cited 2 times on pages 55 and 130.
- SIGILLITO, V. G.; WING, S. P.; HUTTON, L. V.; BAKER, K. B. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, v. 10, n. 3, p. 262–266, 1989. Cited on page 124.
- SOSSA, H.; GUEVARA, E. Efficient training for dendrite morphological neural networks. *Neurocomputing*, Elsevier, v. 131, p. 132–142, 2014. Cited on page 18.
- SPALL, J. C. *Introduction to stochastic search and optimization: estimation, simulation, and control*. [S.l.]: John Wiley & Sons, 2005. v. 65. Cited on page 61.
- SPRINGENBERG, J. T.; DOSOVITSKIY, A.; BROX, T.; RIEDMILLER, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. Cited on page 140.
- SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of Machine Learning research*, JMLR.org, v. 15, n. 1, p. 1929–1958, 2014. Cited on page 77.
- STEINBERG, S. A. *Lattice-ordered rings and modules*. [S.l.]: Springer, 2010. Cited 5 times on pages 32, 33, 35, 36, and 37.
- STERNBERG, S. R. Grayscale morphology. *Computer Vision, Graphics, and Image Processing*, v. 35, n. 3, p. 333 – 355, 1986. ISSN 0734-189X. Cited on page 16.
- STREET, W. N.; WOLBERG, W. H.; MANGASARIAN, O. L. Nuclear feature extraction for breast tumor diagnosis. In: *Biomedical image processing and biomedical visualization*. [S.l.]: International Society for Optics and Photonics, 1993. v. 1905, p. 861–870. Cited on page 123.
- STROBL, C.; MALLEY, J.; TUTZ, G. An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological Methods*, American Psychological Association, v. 14, n. 4, p. 323, 2009. Cited on page 127.
- SUSSNER, P. Morphological perceptron learning. In: IEEE. *Proceedings of the 1998 IEEE International Symposium on Intelligent Control (ISIC) held jointly with IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA) Intell.* [S.l.], 1998. p. 477–482. Cited 5 times on pages 17, 42, 88, 90, and 97.

_____. A relationship between binary morphological autoassociative memories and fuzzy set theory. In: IEEE. *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*. [S.l.], 2001. v. 4, p. 2512–2517. Cited on page 87.

_____. A fuzzy autoassociative morphological memory. In: IEEE. *Proceedings of the International Joint Conference on Neural Networks, 2003*. [S.l.], 2003. v. 1, p. 326–331. Cited on page 87.

_____. Generalizing operations of binary autoassociative morphological memories using fuzzy set theory. *Journal of Mathematical Imaging and Vision*, Springer, v. 19, n. 2, p. 81–93, 2003. Cited on page 87.

SUSSNER, P.; CAMPIOTTI, I. Extreme learning machine for a new hybrid morphological/linear perceptron. *Neural Networks*, Elsevier, v. 123, p. 288–298, 2020. Cited 11 times on pages 19, 20, 39, 86, 94, 107, 108, 120, 121, 127, and 137.

SUSSNER, P.; ESMI, E. Morphological perceptrons with competitive learning: Lattice-theoretical framework and constructive learning algorithm. *Information Sciences*, Elsevier, v. 181, n. 10, p. 1929–1950, 2011. Cited 9 times on pages 37, 38, 41, 42, 86, 91, 97, 99, and 120.

SUSSNER, P.; ESMI, E.; JARDIM, L. G. A submethod interval associative memory with competitive learning. In: SPRINGER. *International Fuzzy Systems Association World Congress*. [S.l.], 2019. p. 643–654. Cited 2 times on pages 39 and 88.

SUSSNER, P.; ESMI, E. L. Constructive morphological neural networks: some theoretical aspects and experimental results in classification. In: *Constructive neural networks*. [S.l.]: Springer, 2009. p. 123–144. Cited 3 times on pages 18, 19, and 92.

_____. An introduction to morphological perceptrons with competitive learning. In: IEEE. *2009 international joint conference on neural networks*. [S.l.], 2009. p. 3024–3031. Cited 6 times on pages 18, 86, 88, 90, 97, and 120.

_____. A submethod interval associative memory with competitive learning. *International Fuzzy Systems Association*, Springer, v. 19, n. 5, p. 643–654, 2019. Cited on page 16.

SUSSNER, P.; ESMI, E. L.; VILLAVERDE, I.; GRAÑA, M. The kosko submethod fuzzy associative memory (KS-FAM): Mathematical background and applications in computer vision. *Journal of Mathematical Imaging and Vision*, Springer, v. 42, n. 2-3, p. 134–149, 2012. Cited 2 times on pages 87 and 136.

SUSSNER, P.; SCHUSTER, T. Interval-valued fuzzy morphological associative memories: Some theoretical aspects and applications. *Information Sciences*, Elsevier, v. 438, p. 127–144, 2018. Cited on page 16.

SUSSNER, P.; VALLE, M. E. Implicative fuzzy associative memories. *IEEE Transactions on Fuzzy Systems*, IEEE, v. 14, n. 6, p. 793–807, 2006. Cited 2 times on pages 16 and 87.

_____. Recall of patterns using morphological and certain fuzzy morphological associative memories with applications in classification and prediction. In: IEEE. *2006 IEEE International Conference on Fuzzy Systems*. [S.l.], 2006. p. 209–216. Cited 2 times on pages 16 and 87.

- VALDIVIEZO-NAVARRO, J.-C.; URCID-SERRANO, G. Hyperspectral endmember detection based on strong lattice independence. In: *Applications of Digital Image Processing XXX*. [S.l.]: International Society for Optics and Photonics, 2007. v. 6696, p. 669625. Cited on page 16.
- VALLE, M. E.; SOUZA, A. C. de. On the recall capability of recurrent exponential fuzzy associative memories based on similarity measures. *Mathware and Soft Computing Magazine*, v. 22, p. 33–39, 2015. Cited on page 39.
- VALLE, M. E.; SUSSNER, P. Fuzzy morphological associative memories based on uninorms. In: IEEE. *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. [S.l.], 2008. p. 1582–1589. Cited on page 87.
- _____. A general framework for fuzzy morphological associative memories. *Fuzzy Sets and Systems*, Elsevier, v. 159, n. 7, p. 747–768, 2008. Cited 4 times on pages 16, 39, 41, and 87.
- VALLE, M. E.; SUSSNER, P.; GOMIDE, F. Introduction to implicative fuzzy associative memories. In: IEEE. *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*. [S.l.], 2004. v. 2, p. 925–930. Cited on page 87.
- VILLAVERDE, I.; GRAÑA, M.; D'ANJOU, A. Morphological neural networks for localization and mapping. In: IEEE. *2006 IEEE International Conference on Computational Intelligence for Measurement Systems and Applications*. [S.l.], 2006. p. 9–14. Cited on page 135.
- VINCENT, P.; LAROCHELLE, H.; LAJOIE, I.; BENGIO, Y.; MANZAGOL, P.-A. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, v. 11, n. Dec, p. 3371–3408, 2010. Cited on page 81.
- VISWANATHAN, T. Generalization of Hölder's theorem to ordered modules. *Canadian Journal of Mathematics*, Cambridge University Press, v. 21, p. 149–157, 1969. Cited on page 35.
- WAN, L.; ZEILER, M.; ZHANG, S.; CUN, Y. L.; FERGUS, R. Regularization of neural networks using dropconnect. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2013. p. 1058–1066. Cited on page 140.
- WENG, C.; YU, D.; WATANABE, S.; JUANG, B.-H. F. Recurrent deep neural networks for robust speech recognition. In: IEEE. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.], 2014. p. 5532–5536. Cited on page 55.
- WIDROW, B.; HOFF, M. E. *Adaptive switching circuits*. [S.l.], 1960. Cited on page 53.
- WIDROW, B.; LEHR, M. A. 30 years of adaptive neural networks: perceptron, MADALINE, and backpropagation. *Proceedings of the IEEE*, IEEE, v. 78, n. 9, p. 1415–1442, 1990. Cited on page 52.
- WINTER, C. R.; WIDROW, B. Madaline rule II: a training algorithm for neural networks. In: *Second Annual International Conference on Neural Networks*. [S.l.: s.n.], 1988. p. 1–401. Cited on page 52.

XU, C.; LU, C.; LIANG, X.; GAO, J.; ZHENG, W.; WANG, T.; YAN, S. Multi-loss regularized deep neural network. *IEEE Transactions on Circuits and Systems for Video Technology*, IEEE, v. 26, n. 12, p. 2273–2283, 2015. Cited 2 times on pages 140 and 144.

YUILLE, A. L.; RANGARAJAN, A. The concave-convex procedure (CCCP). In: *Advances in neural information processing systems*. [S.l.: s.n.], 2002. p. 1033–1040. Cited on page 106.

ZAMORA, E.; SOSSA, H. Dendrite morphological neurons trained by stochastic gradient descent. *Neurocomputing*, v. 260, p. 420 – 431, 2017. ISSN 0925-2312. Cited 7 times on pages 18, 19, 20, 101, 102, 114, and 121.