UNIVERSITÀ DEGLI STUDI DI SIENA

Dipartimento di Ingegneria dell'Informazione
e Scienze Matematiche

On the Integration of Logic and Learning

Francesco Giannini

Ph.D Thesis in Information Engineering and Science

Supervisor

Prof. Marco Maggini

Siena, 2019

Acknowledgements

I would firstly like to thank Prof. Marco Maggini for everything he taught me during my Ph.D program. For all the useful suggestions and the pleasant chats, for always being ready to go into details. I thank Prof. Marco Gori very much for having always encouraged me, for the many exchanges of ideas and theories, and above all for introducing me, with my "logic", to the world of artificial intelligence. A special mention goes to Prof. Michelangelo Diligenti, for being always available to help and for all the interests he shared with me. Both for the intense days of work and those on the top of the mountains. In this sense he was much more than a teacher to me and I can only thank him very much. I also thank Prof. Franco Scarselli, Monica Bianchini and Stefano Melacci for the useful discussions and suggestions.

I would also like to thank all the Ph.D students that shared with me this intense experience over these years. I made several new good friends and together we shared both the joys and the moments of frustration. In particular, I thank Giuseppe for the endless philosophical discussions on the mind, Andrea for the cinema, Dario for the art, Sara for always encouraging me, as well as Vincenzo, Alessandro, Matteo, Simone, Paolo, and all the other ones that have made working together a pleasure.

Finally, I thank my family for everything it has done for me in these years. In particular my parents, for giving me the opportunity to continue my studies and pursue my dreams, my grandparents and my uncles for all their support and affection. And above all I thank Valentina, for putting up with me during the most difficult periods and for being with me in any new challenges.

Abstract

A key point in the success of machine learning, and in particular deep learning, has been the availability of high-performance computing architectures allowing to process a large amount of data. However, this potentially prevents a wider application of machine learning in real world applications, where the collection of training data is often a slow and expensive process, requiring an extensive human intervention. This suggests to look at possible ways to overcome this limitation, for instance injecting prior knowledge into a learning problem to express some desired behaviors for the functions to be learned.

In this thesis, we consider the case of prior knowledge expressed by means of first-order logic formulas to be integrated into a learning problem. In particular, at first the formulas are converted into real-valued functions by means of t-norm fuzzy logic operators. Thereafter, a loss component (a constraint) is assigned to any function representing a formula and all these components are aggregated (e.g. summed) together with other possible loss components, e.g. a regularization term or some loss components associated to supervisions, if they are available for the functions to be learned. Both the functional representation of a formula and the mapping into a loss component have been investigated, and some theoretical results are discussed to get an insight on how to bring some benefits for different learning schemata. In particular we define a fragment of Łukasiewicz logic that guarantees to yield convex functional constraints given any knowledge base made of first-order logic formulas. The convexity of these constraints is exploited to formulate collective classification as a quadratic optimization problem and some experimental results are discussed. In addition, we extend classic Support Vector Machines with logical constraints, still preserving quadratic programming resolution. Since formulas may be logically depending on each other, some of the constraints may turn out to be unnecessary with respect to the learning process. This suggests to generalize the notion of support vector to support constraint, and we provide both logical and algebraic criteria to determine the constraints that are unnecessary. Finally we present LYRICS, a general interface implemented in TensorFlow to integrate both deep learning architectures and a first-order logic representation of knowledge for a learning problem. In particular, we show several learning tasks that may be addressed in LYRICS, with a special discussion for the case of visual generation.

Contents

Acknowledgements i
Abstract ii

1 Introduction 3
1.1 Motivation 5
1.2 Thesis Summary 7
1.2.1 Major Contributions of the Thesis 7
1.3 Structure of the Thesis 8

2 Logic and Machine Learning Foundations 11
2.1 Boolean Logic 12
2.1.1 Boolean Predicate Calculus 13
2.2 Fuzzy Logic 15
2.2.1 Functional Representation of Fundamental Fuzzy Logics 19
2.3 Fuzzy Aggregation Functions 21
2.3.1 Generated Archimedean T-Norms 24
2.4 Machine Learning 29
2.4.1 Tasks 30
2.4.2 Types of Algorithms 32
2.4.3 Some Models 34
2.4.4 ML as Learning from Constraints 38

3 Learning and Logical Reasoning 41
3.1 Related Works 41

3.2 Markov Logic Networks 46
3.3 Probabilistic Soft Logic 49
3.4 Semantic–Based Regularization 53
3.5 Logic Tensor Networks 55

4 Convex Logical Constraints 57
4.1 The Convex Łukasiewicz Fragment 60
4.1.1 Propositional Łukasiewicz Logic 61
4.1.2 Convex McNaughton Functions 64
4.1.3 Notes on Expressiveness 67
4.2 Fuzzy Aggregation and Loss Functions 70
4.2.1 Loss Functions by T-Norms Generators 70

5 Learning with Convex Logical Constraints 75
5.1 Kernel Machines 76
5.1.1 SVMs with Logical Constraints 76
5.1.2 Experimental Results 83
5.2 Support Logical Constraints 87
5.2.1 Active Constraints 88
5.2.2 Unnecessary Constraints 96
5.3 Collective Classification 102
5.3.1 Experimental Results 104
5.4 Extending Probabilistic Soft Logic 106

6 LYRICS 109
6.1 The Declarative Language 110
6.2 From Logic to Learning 112
6.3 Example Driven Presentation of LYRICS 117
6.4 Generative Learning and Visual Translation 125
6.4.1 Expressing an Adversarial Setting by FOL 126
6.4.2 Experimental Results 128

7 Deep Logic Models 137

8 Conclusions and Future Work **143**
8.1 Concluding Remarks 143
8.2 Future Work 145

List of Publications **149**

Bibliography **151**

List of Figures

5.1 Solution exploiting only few supervisions 84
5.2 Solutions exploiting logical constraints on unsupervised data . 85
5.3 Optimal solutions under unnecessary constraints 93
5.4 Collective classification accuracy 105

6.1 LYRICS architecture 113
6.2 From FOL to computational graph 113
6.3 LYRICS: Semi–supervised Learning 118
6.4 LYRICS: Collective classification 120
6.5 LYRICS: Missing Features 124
6.6 Generation of *next* and *previous* digits 130
6.7 Face Gender Translation: male to female 133
6.8 Face Gender Translation: female to male 133
6.9 Face Gender Translation: male/female to eyeglasses 134

7.1 Representation of a DLM graphical model 139

List of Tables

2.1 Truth tables of Classical Logic 13
2.2 Truth Functions for the Fundamental Fuzzy Logics 18

4.1 Łukasiewicz logic truth functions 62

5.1 Knowledge base of the Winston benchmark 86
5.2 F1 score for the Winston benchmark 87

Chapter 1

Introduction

There is a long tradition in the literature relating *Mathematical Logic* to *Artificial Intelligence* (AI) tasks. On the one hand, the former provides a rigorous and mathematical tool to investigate the soundness of reasoning. On the other hand AI focuses on the capacity of machines to mimic "cognitive" functions, generally associated to an intelligent behavior, such as learning and problem solving. In particular, several logical systems have been considered to investigate formal environments where, by means of a set of axioms, some facts are fixed and inference obeys to one or more explicit rules. Modifying the axioms and the inference rules, different aspects of real behaviors can be modeled and rigorously investigated. The most known example is given by *Boolean Logic* where any proposition is assigned a truth value, either 0 for "false" or 1 for "true". However, in modeling a real world environment, we could deal with partial or imprecise information that is not suitable to be interpreted as absolutely true nor absolutely false. This is a reason why, *many-valued logics*, and in particular *fuzzy logics*, have been introduced to represent and manipulate a set of truth degrees that in case of fuzzy logic coincides with the unit interval $[0, 1]$. This is a reason why it has been applied to many fields, from control theory to artificial intelligence and in particular in this thesis, we adopt a fuzzy logic representation of knowledge to inject logical formulas into the continuous optimization problem at the basis of the learning process.

In the last few years, machine learning techniques have reached amazing results in many Artificial Intelligence tasks. This is mostly due to the capability of deep learning to jointly learn feature representations and classification models, especially when dealing with high dimensional input patterns. On the other hand, a real intelligent behavior of an agent acting in a complex environment is likely to require some kind of higher-level symbolic inference. Therefore, there is a clear need for the definition of a general and tight integration between low-level tasks, processing sensorial data that can be effectively elaborated using deep learning techniques, and logic reasoning that allows humans to take decisions in complex environments.

This thesis lays at the intersection of two fields, machine learning and mathematical (fuzzy) logic. In particular, we present a general framework to integrate these two disciplines both providing theoretical results to be exploited into some learning schema and implementing in TensorFlow a general interface, called LYRICS. LYRICS is a software realizing the integration of deep neural networks and logical reasoning allowing the full expressiveness of first-order logic. Combining machine learning and logical reasoning has been proposed by several authors with different approaches in the literature. In general, formal logic allows us to express the knowledge about a learning problem by manipulable symbols occurring in formulas. However, these symbols may be chosen according to different language orders and formulas may be integrated into a learning problem according to different strategies and exploiting possible structures of the available information. On the other hand, the majority of machine learning techniques rely on the general paradigm of defining a loss function measuring the performance of a certain model on a certain task, and minimizing the cost in order to learn the best model parameters. This is a reason why the framework we propose is based on the conversion of logical formulas into loss components (constraints expressing the distance from satisfaction of the formulas) to be optimized with respect to the model parameters. This conversion may be carried out in several ways, however the main aspects are basically two. The first one is how to give a functional representation to formulas, the second one is how to assign to each functional a loss component. The way we decide to carry out these two steps gives rise to different loss functions, some of which may be easier to be optimized by a learning algorithm. This suggests to analyze the algebraic properties of functions corresponding to formulas and in particular we show that a fragment of Łukasiewicz logic yields convex constraints. Since the result is about logic, it may be exploited to get benefits in different learning schemata integrating prior knowledge by logical formulas.

With LYRICS, we propose a TensorFlow library automatically implementing the conversion of a high-level representation given by logical formulas into loss functions and an integration with task-oriented deep learning architectures. On the one hand LYRICS allows us to formulate a learning problem as a first-order logic (FOL) model where variable domains, FOL functions and predicate functions are declared. On the other hand, providing real data and machine learning models to implement the involved functions, the learning problem is defined by converting and aggregating all the constraints (formulas) into an overall loss function to be optimized.

1.1 Motivation

In spite of the amazing results obtained by deep learning in many applications, a real intelligent behavior of an agent acting in a complex environment is likely to require some kind of higher-level symbolic inference. Therefore, there is a clear need for the definition of a general and tight integration between low-level tasks, processing sensorial data that can be effectively elaborated using deep learning techniques, and logic reasoning that allows humans to take decisions in complex environments. The success of deep learning relies on the availability of a large amount of supervised training data, however in real world applications, differently from ad hoc benchmarks, data can be partially labeled or contain labeling mistakes making the application of deep architectures generally difficult. On the other hand, concerns on what learning really means raise: how much deep learning simply relies on remembering previously seen patterns? How robust is such a system in an adversarial or uncontrolled environment, where outliers are either unpredictable or even forged to fool the trained system? These concerns have shifted the attention onto automatic feature development for supervised learning without the injection on any prior knowledge at the problem at hand. The introduction of prior-knowledge into the learning process is a fundamental step in overtaking these limitations. First, it does not require the training process to induce the rules from the training set, therefore reducing the number of required training data. Secondly, the use of prior-knowledge can be used to express the desired behavior of the learner on any input, providing benefits in a semi-supervised environment. In addition, a high level declarative mechanism can be an easier and more natural way to express the architecture of the learning task, instead of relying on hand-crafted cost functions.

The integration of deep learning and logic reasoning is still an open-research problem and it is considered to be the key for the development of real intelligent agents. This integration gives rise to several related issues and this thesis discusses possible solutions in this respect. For instance, we focus on the way some prior-knowledge expressed by logical formulas may be converted into functional constraints and injected into a learning scheme. On the other hand, an optimization problem can be solved with efficient algorithms in the case it is convex, or even better if it is quadratic. In particular, convex optimization is not affected by the presence of local minima, indeed any local optimum is also global and it guarantees the existence (and unicity in the strong convex case) of an optimal solution under opportune hypothesis, e.g. the KKT-conditions that are both necessary and sufficient for the convex programming case. This is a reason why we investigated the possible conversions of formulas into loss components and we provide a Łukasiewicz logic fragment allowing us to map first-order logic formulas into convex functional constraints. Moreover, in a learning from constraints problem, several constraints may be involved in the optimization process, but some of them may turn out to be unnecessary. In the specific case of logical constraints, where some consequence relations may hold among the involved formulas, we provide some analytical criteria to determine which are the unnecessary constraints for a certain learning problem. As a result, the unnecessary constraints may be cut off and the number of constraints to be enforced can be reduced still getting the same feasible solutions for the optimization process.

As we already noticed, a machine learning problem may be generally defined by formulating a loss function to be minimized, and logical formulas may express the knowledge about the problem and therefore be converted into additional loss components. In order to implement an environment where a learning problem is totally defined by first-order logic formulas and any unknown function to be learned can be represented by an opportune machine learning model, we present LYRICS. In LYRICS, once all the learning objects are defined, any formula is automatically converted into a loss component according to a certain fuzzy logic semantics. This allows us to formulate a learning problem by declaring all the available information for the task by means of constraints in a high-level language, i.e. by first-order logic formulas, and then parsing them into loss components. In this respect, the above mentioned theoretical results can be applied in LYRICS, simply specifying how the formulas have to be mapped into functional constraints.

1.2 Thesis Summary

This thesis presents some novel results on the way a set of FOL formulas can be injected into a learning problem. In particular, the FOL rules are converted into functional constraints and aggregated into an overall loss function. In this respect, we provide a Łukasiewicz logic fragment yielding convex functional constraints and we show that it is maximal to preserve convexity. Expressing formulas in this fragment allows us to extend classic Support Vector Machines still preserving quadratic optimization and some experimental results are discussed. The consequence relation among formulas can make some constraints unnecessary with respect to the learning problem and we provide both logical and algebraic criteria to determine which formulas may be cut off without influencing the set of possible solutions. Finally, the theoretical results can be exploited in LYRICS, where several machine learning tasks may be formulated in a constraint-based environment with full first-order logic expressiveness.

1.2.1 Major Contributions of the Thesis

The contributions of this thesis may be summarized as follows.

1. Identification of a logical fragment of Łukasiewicz logic coinciding with the class of all the McNaughton functions that are convex.
   Based on [52].

2. Definition of collective classification problems as quadratic optimization exploiting the proposed convex fragment of Łukasiewicz logic.
   Based on [51].

3. Extension of Support Vector Machines by logical constraints still preserving quadratic optimization.
   Based on [50].

4. Generalization of the notion of support vector to support constraint. In particular, some criteria to determine if a certain constraint is unnecessary are provided.

5. Development of LYRICS, a TensorFlow implementation extending Semantic Based Regularization with first-order logic functions and providing a high-level language to express the constraints.
   Based on [98].

6. Introduction of Deep Logic Models, a class of graphical models integrating symbolic and sub-symbolic learning. However, this framework is still under investigation and we do not provide any experimental result in this thesis.
   Based on [97].

1.3 Structure of the Thesis

The thesis is organized as follows.

Chapter 2 recalls the fundamentals of mathematical logic and machine learning theories and introduces some terminology that will be exploited in the following chapters. For what concerns logic, we introduce the well-known Boolean logic both in propositional and first-order case, and Fuzzy logic that provides an extension for truth evaluation of formulas in the unit interval. In addition, we introduce general fuzzy aggregation functions and in particular we discuss the case of generated archimedean t-norms. For what concerns machine learning, we give an overview of the main constituents of this field. In particular, we describe some common tasks, algorithms and models, reporting some examples and emphasizing how machine learning can be related to general learning from constraints.

Chapter 3 discusses some related literature and presents some frameworks where the integration of machine learning and logical reasoning is realized. In particular, we present in more detail Markov Logic Networks, Probabilistic Soft Logic, Semantic-Based Regularization and Logic Tensor Network. However, a wide variety of frameworks are described in order to give a general overview of the relevant approaches considered by different authors. In addition, we briefly introduce Deep Logic Models, a novel architecture integrating probabilistic logic and deep neural networks.

Chapter 4 presents the main theoretical results of the thesis. To start, we describe the approach we adopt to convert formulas into continuous functions and thereafter into loss functions. To this regard, we identify a fragment of Łukasiewicz logic guaranteeing convexity for the functional representations, but some others conversions are discussed. In particular, since a special class of t-norms have a representation in terms of unary functions called generators, we investigate the use of generators to produce different loss functions.

Chapter 5 discusses how the concave fragment of Łukasiewicz logic may be exploited in different learning schemata to inject convex functional constraints. For instance, we show how to extend Support Vector Machines still preserving quadratic programming and we show some experimental results. As a consequence, the notion of support vector can be refined to support constraint and we show how to determine the constraints that are not necessary for an optimization process, providing some theoretical characterization of unnecessary constraints. In addition, we show that the proposed logical fragment yields an expressiveness extension of Probabilistic Soft Logic.

Chapter 6 presents LYRICS, a general interface to integrate first-order logic and machine learning architectures. Firstly, we define the LYRICS language providing some examples on how to implement a learning environment and we describe how the constraints are automatically mapped into TensorFlow computational graphs. Secondly, we implement some examples of learning and reasoning tasks that can be carried out in LYRICS in order to enlighten its versatility. Finally, we discuss the case of a generative (adversarial) setting showing the simplicity of the proposed framework, e.g. in defining generally complex loss functions associated to Generative Adversarial Networks.

Chapter 7 briefly introduces Deep Logic Models, a framework allowing the integration of deep learning architectures with probabilistic logic inference in graphical models. The model is considered for the purpose of overcoming some limitations of existing frameworks combining symbolic and sub-symbolic approaches.

Chapter 8 outlines a discussion and concluding summary of the research presented in the thesis. The major findings of the thesis are discussed with an overall summary of the contributions given. Further, we discuss some possible extensions for the presented results and possible areas of future work.

Chapter 2

Logic and Machine Learning Foundations

The work presented in this thesis is fundamentally based on two disciplines, *Machine Learning* and *Mathematical Logic*, whose basic notions are described in this chapter. Machine learning can be seen as the study of algorithms and statistical models that computer systems may implement to progressively improve their performance on a specific task. In particular, it concerns the construction of a mathematical model from some "training data", in order to make predictions on unseen data without being explicitly programmed to perform the task. On the other hand, mathematical logic is a subfield of mathematics focused on the application of formal logic to mathematics. In particular, it concerns the study of the consequence relation among propositions and the way truth preserves by reasoning. Indeed the strengths of combining logic with learning settings are multiple. For instance, logic offers different formal languages to express the available knowledge for a learning setting according to the granularity of the considered expressiveness, e.g. propositional logic and first–order logic. In addition, allowing the embedding of a reasoning process into a learning problem generally outperforms the performance on a certain task, reducing the number of required supervised data and reinforcing the learnable facts that are coherent with the logic rules.

The chapter is organized as follows. In the first three sections we present the main logical frameworks we will deal with as well as some terminology and fundamental results. We start in Sec. 2.1 introducing *boolean logic* both in the propositional and first–order case. The knowledge expressed in a learning problem generally consists of a set of boolean formulas, however in order to deal with continuous values it may be suitable to consider a fuzzy logic. In Sec. 2.2 we introduce the basic notions of t-norm fuzzy logics as well as the theorems characterizing a functional representation of the fundamental fuzzy logics. In addition, in Sec. 2.3 we extend the investigation on fuzzy connectives to the wide class of *fuzzy aggregation functions* allowing to aggregate several real values. Finally in Sec. 2.4 we introduce the field of machine learning, sketching some of the most common tasks it faces, algorithms and learning models.

2.1 Boolean Logic

Since ancient times, *mathematical logic* has been considered as the rigorous discipline studying the soundness of reasoning and the notion of consequence among propositions. In particular, we are interested in investigating the way truth preserves and, in this sense, the prime example is given by *Classical (Boolean) Logic* (CL) where any proposition has associated either the truth value 1 (true) or 0 (false). Depending on the granularity in the theory, a logic can be defined by a *propositional* or a *higher-order* language.

The syntax of Boolean propositional logic [39, 138] is built from a (possible infinite) set of *propositional variables* $p_1, \ldots, p_n, \ldots$; two propositional *constants* $\bar{0}, \bar{1}$ denoting the False and True proposition respectively; and the logical *connectives*: $\wedge$ *conjunction*, $\vee$ *disjunction*, $\neg$ *negation*, $\rightarrow$ *implication*, $\leftrightarrow$ *equivalence*. These elements are combined according to the following inductive definition to build the set of formulas:

- propositional variables and constants are formulas;
- if φ, ψ are formulas, then $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \rightarrow \psi), (\varphi \leftrightarrow \psi)$ are formulas.

Any formula represents a proposition whose truth value has to be evaluated. Formally, a (*truth*) *evaluation* is a mapping from the set of propositional variables to $\{0, 1\}$. As a result, the truth value associated to any formula is defined, according to truth functionality, by the truth functions associated to the logical connectives occurring in the formula. In Tab. 2.1 the truth tables of the logical connectives are reported, whereas the logical constants $\bar{0}, \bar{1}$ are always evaluated in $0, 1$ respectively. For clarity, with a little abuse of notation, both the logical connectives and their corresponding semantic operations are denoted by the same symbols.

p, q	$\neg p$	$p \wedge q$	$p \vee q$	$p \to q$	$p \leftrightarrow q$
0, 0	1	0	0	1	1
0, 1	1	0	1	1	0
1, 0	0	0	1	0	0
1, 1	0	1	1	1	1

Table 2.1: *Truth tables of classical logic connectives.*

Syntax and semantics are related by the well-known **Soundness and Completeness Theorem**, where *theorems* (provable formulas from CL axioms) and *tautologies* (formulas that are evaluated in 1 for any truth evaluation) are shown to coincide. However, a detailed explanation of this topic is out of scope and we recommend e.g. [39, 138] for the interested reader.

We introduced Boolean Logic in order to define some fundamental terminology about logic that is used in the next chapters. In particular:

- a *literal* is a propositional variable or its negation;
- a *Horn Clause* is a formula $l_1 \wedge \ldots \wedge l_n \to l$, for some literals $l_1, \ldots, l_n, l$;
- a *CNF* (Conjunctive Normal Form) is a formula expressed as a conjunction of disjunctions of literals;
- a *DNF* (Disjunctive Normal Form) is a formula expressed as a disjunction of conjunctions of literals.

It is a well-known result [39] that in CL any formula can be equivalently rewritten in CNF (and dually in DNF). This is an immediate consequence of the double negation law ($\neg\neg\phi \leftrightarrow \phi$) and the following properties hold in CL:

De Morgan:	$\neg(\alpha \wedge \beta) \leftrightarrow \neg\alpha \vee \neg\beta$	$\neg(\alpha \vee \beta) \leftrightarrow \neg\alpha \wedge \neg\beta$
Distr. Laws:	$\alpha \wedge (\beta \vee \gamma) \leftrightarrow (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$	$\alpha \vee (\beta \wedge \gamma) \leftrightarrow (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$

2.1.1 Boolean Predicate Calculus

Propositional logic does not allow us to express any structure in atomic formulas. This is a reason why *First–Order Logic* (FOL) is more suitable for settings where some relational knowledge among the objects of a certain domain can be expressed. Indeed, while propositional logic deals with simple declarative propositions, FOL additionally covers predicates, functions and quantification. For instance, consider the two sentences "Socrates is a philosopher" and "Plato is a philosopher". In a propositional language, these sentences are viewed as being unrelated and have to be denoted by two different variables such as p and q. On the other hand, the predicate "is a philosopher" occurs in both sentences, which have a common structure that could be denoted, for instance, by *philosopher(Socrates)* and *philosopher(Plato)*.

Formally, a predicate language consists of a non-empty set of *predicates* (also called relations) $p^{(a_p)}, q^{(a_q)}, \ldots$ each together with a positive natural number $a_p, a_q > 0$ (its arity); a (possibly empty) set of functions $f^{(a_f)}, g^{(a_g)}, \ldots$ each together with its arity; a (possibly empty) set of object constants $c, d, \ldots$ and an infinite set of variables $x, y, z \ldots$. Finally, logical connectives $(\wedge, \vee, \ldots)$ and $\bar{0}, \bar{1}$ are defined as in the propositional case, while $\forall, \exists$ denote the universal and existential quantifiers, respectively. The set of terms will refer to objects in a certain domain and it is inductively defined as:

- variables and constants are terms;
- if $t_1, \ldots, t_n$ are terms and $f^{(n)}$ is an n–ary function, then $f^{(n)}(t_1, \ldots, t_n)$ is a term.

In first–order logic, the set of formulas is defined upon predicates as:

- $\bar{0}, \bar{1}$ are formulas. If $t_1, \ldots, t_n$ are terms and $p^{(n)}$ is an n–ary predicate, then $p^{(n)}(t_1, \ldots, t_n)$ is a formula, said *atomic* formula;
- if φ, ψ are formulas, then $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi), (\varphi \to \psi), (\varphi \leftrightarrow \psi)$ are formulas;
- if x is a variable and φ is a formula, then $(\forall x)\,\varphi, (\exists x)\,\varphi$ are formulas.[1]

The semantics of a predicate language is related to *structures*, where a structure $\mathbf{M} = (M, (r_p)_p, (r_f)_f, (m_c)_c)$ is given by a non-empty domain M[2],

[1]In the following we will omit brackets in formulas and arity symbols when clear from the context.

[2]Since we are interested in real-world applications, we will consider the case of finite domains only.

for each object constant c, $m_c \in M$, for each n–ary function f, $r_f : M^n \to M$ and for each n–ary predicate p, $r_p \subseteq M^n$. In addition, also *many-sorted* first–order logics, allowing variables to have different sorts with different domains, can be considered. However many-sorted first–order logic can be reduced to single-sorted first–order logic in case of finite number of sorts [40]. In order to assign a truth value to any formula, we need to associate an element in a domain to any term. Given a certain structure $\mathbf{M}$, an $\mathbf{M}$–evaluation v is inductively defined such that, for every variable x, $v(x) \in M$; for every n–ary function f and terms $t_1, \ldots, t_n$, $v(f(t_1, \ldots, t_n)) = r_f(v(t_1), \ldots, v(t_n))$.

Finally, a truth value is associated to any formula with respect to an M–evaluation v by:

$$\begin{array}{ccc} p(t_1, \ldots, t_n) & \longrightarrow & (v(t_1), \ldots, v(t_n)) \in r_p ? 1 : 0 \\ \neg\varphi, \varphi \circ \psi \text{ with } \circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow\} & \longrightarrow & \text{as in the propositional case.} \\ \forall x \varphi(x) & \longrightarrow & \min_x(\varphi(x)) \\ \exists x \varphi(x) & \longrightarrow & \max_x(\varphi(x)) \end{array}$$

where the expression $x?y : z$ is called the *conditional* or *ternary* operator, returning y if x is true and z otherwise.

In the next chapters, we will refer to the following nomenclature.

- Given a set of formulas T, $\mathbf{M}$ is said a *model* of T if every formula in T is evaluated in 1 for any $\mathbf{M}$–evaluation.
- A formula is said to be in *prenex normal form* (pnf) if it is in the form $Q_1, \ldots, Q_n\varphi$ where $Q_1, \ldots, Q_n \in \{\forall, \exists\}$ and φ does not contain any quantifier.

Theorem 2.1. *Any formula of classical (both in propositional and first–order) logic has an equivalent pnf [39].*

2.2 Fuzzy Logic

In Classical Logic, we can only deal with two truth values, 0 for false and 1 for true. The main difference with both many-valued and fuzzy logics is to extend the set of truth values. In fuzzy logic the real unit interval $[0, 1]$ is taken as set of truth values, where 0 denotes the absolute false and 1 the absolute true.

Fuzzy logics have widely been investigated in the literature by several authors with different intentions. In this section, we introduce the basic notions of propositional[3] fuzzy logics and the three main continuous fuzzy logics, e.g. Gödel, Łukasiewicz and Product. Fuzzy logics enjoy truth functionality and, according to [62], they can be defined upon a certain *t-norm* (triangular norm) representing a continuous extension of boolean conjunction. All the chosen truth functions of connectives must be a generalization of classical two-valued logic, hence when restricted to crisp values $\{0, 1\}$ they have to be equal to their corresponding booleans.

Definition 2.1 (t-norm)**.** *An operation* $\otimes : [0, 1]^2 \to [0, 1]$ *is a t-norm if and only if for every* $x, y, z \in [0, 1]$*:*

$$x \otimes y = y \otimes x, \quad x \otimes (y \otimes z) = (x \otimes y) \otimes z, \quad x \otimes 1 = x, \quad x \otimes 0 = 0,$$
$$x \leq y \longrightarrow x \otimes z \leq y \otimes z, \qquad x \leq y \longrightarrow z \otimes x \leq z \otimes y \ .$$

$\otimes$ *is a continuous t-norm if it is continuous as function.*

Example 2.1. *Some of the most prominent examples of t-norms are:*

Gödel: $x \otimes_G y = \min\{x, y\}$

Łukasiewicz: $x \otimes_Ł y = \max\{0, x + y - 1\}$

Product: $x \otimes_\Pi y = x \cdot y$

Drastic: $$x \otimes_D y = \begin{cases} x & \textit{if y=1} \\ y & \textit{if x=1} \\ 0 & \textit{otherwise} \end{cases}$$

Gödel, Łukasiewicz and Product t-norms are referred as the fundamental t-norms because all the continuous t-norms can be characterized [106] with respect to them by ordinal sums[4]. The Drastic t-norm is not continuous, however it is interesting to notice that this is the least t-norm with respect to the partial order among t-norms defined as follows. This order will play a

[3]The fuzzy generalization of FOL is not presented in this work since, as we better clarify in Chap. 4, we are mostly interested in the propositional case.

[4]The interested reader is referred to [69] for a definition of ordinal sums for t-norms.

fundamental role in Sec. 4.2, where we will extend the analysis to the wide variety of fuzzy aggregation operators.

$$\otimes_1 \leq \otimes_2 \text{ iff } x \otimes_1 y \leq x \otimes_2 y, \ \forall x, y \in [0,1] \ . \tag{2.1}$$

A symmetric generalization of disjunction to continuous values in $[0,1]$ is given by the notion of t-conorm. This is defined as a t-norm except the identity and annihilating elements are inverted. We will come back to this notion underlining its relation with the t-norm after introducing strong negations.

Given a continuous t-norm $\otimes$, it is definable its corresponding residuum $\Rightarrow$ that generalizes the notion of implication and it is uniquely determined by

$$x \Rightarrow y = \max\{z : x \otimes z \leq y\} \ .$$

This is strictly related to the *law of residuation*

$$x \otimes z \leq y \quad \text{iff} \quad z \leq x \Rightarrow y$$

that, as a special case, yields $x \leq y$ if and only if $(x \Rightarrow y) = 1$.

It is worth noticing that, in any fuzzy logic, a t-norm and its residuum allow us to recover two additional logical connectives related to the order among reals $\leq$, a conjunction and a disjunction, whose truth functions are the min and max operations respectively. As a special case, we note that in Gödel logic the minimum corresponds exactly to the t-norm. These connectives are said *weak* in contrast to t-norms and t-conorms that are called *strong*. The min and max operations, denoted by $\wedge$ and $\vee$ respectively, are defined as:

$$x \wedge y = x \otimes (x \Rightarrow y) \qquad x \vee y = ((x \Rightarrow y) \Rightarrow y) \otimes ((y \Rightarrow x) \Rightarrow x) \ . \tag{2.2}$$

As we already pointed out, the whole fuzzy logic can be determined by the chosen t-norm. In addition, by means of the residuum, it is definable the truth function of a negation by the unary operation $\neg x = x \Rightarrow 0$. Among negations, the Łukasiewicz negation $\neg x = 1 - x$ is called the *standard* negation and plays a special role. Indeed, among the three negations of the fundamental t-norms, this is the only one to be a *strong* negation, namely to be continuous, strictly decreasing and involutive (e.g. $\neg\neg x = x$). Further, each strong negation was shown to be a monotone transformation of the standard negation [145]. Given

	Gödel	Łukasiewicz	Product
$x \otimes y$	$\min\{x, y\}$	$\max\{0, x + y - 1\}$	$x \cdot y$
$x \Rightarrow y$	$x \leq y?1 : y$	$\min\{1, 1 - x + y\}$	$x \leq y?1 : \frac{y}{x}$
$\neg x$	$x = 0?1 : 0$	$1 - x$	$x = 0?1 : 0$
$x \oplus y$	$\max\{x, y\}$	$\min\{1, x + y\}$	$x + y - x \cdot y$
$x \rightarrow y$	$\max\{1 - x, y\}$	$\min\{1, 1 - x + y\}$	$1 - x + x \cdot y$

Table 2.2: *Up to down are reported the truth functions of t-norms, residuum, negation, dual t-conorms and material implication of the fundamental fuzzy logics.*

a t-norm $\otimes$ and a strong negation $\sim$, the function defined as follows is called its $\sim$-*dual t-conorm* (only *dual t-conorm* if $\sim$ is the standard negation).

$$x \oplus y = \sim (\sim x \otimes \sim y)$$

T-conorms are the t-norm counterparts generalizing the notion of boolean disjunction to the real unit interval. It is worth noticing that it is not possible to internally define the dual t-conorm in a t-norm fuzzy logic without a strong negation. However, some studies on extending fuzzy logics by an involutive negation have been considered in the literature [17, 41, 43]. On the other hand, a quite large amount of work on structures characterized by a strong negation, t-norm and its dual t-conorm with respect to that negation has been done. Such structures are called *De Morgan triples* and are studied by several authors with different applications [4,47,48,154]. The basic difference between the setting of Mathematical Fuzzy logic and De Morgan triples is the use of the residuated implication and its associated negation. Anyway, in De Morgan triples we can easily recover the *material* implication (here denoted by $\rightarrow$) determined by the t-conorm as:

$$x \rightarrow y = \sim x \oplus y \ .$$

In our analysis, the only strong negation we will take into account is the standard one and in Tab. 2.2 the t-conorms and material implications of the fundamental fuzzy logics are expressed with respect to this negation.

Some further remarks are in order.

- The only continuous residuum is the Łukasiewicz one, see Tab. 2.2.

- Extending a fuzzy logic with an involutive negation allows us to increase its expressiveness and recover the De Morgan law on the t-norm.

- In general, the distributive laws do not hold between a t-norm and its dual t-conorm. For instance, for $x \in (0,1)$ in Product logic:

$$x \otimes (y \oplus z) = xy + xz - xyz \neq xy + xz - x^2yz = (x \otimes y) \oplus (x \otimes z)$$

2.2.1 Functional Representation of Fundamental Fuzzy Logics

In the next chapters, fuzzy logic truth functions will be exploited into a learning schema to represent the satisfaction of certain functional constraints in an optimization problem. Actually, functional representation of propositional fuzzy logic formulas has been studied by different authors, see for instance [3] for a detailed analysis of Gödel, Łukasiewicz and Product logic. The main idea is that the algebra of formulas on n variables of any fundamental fuzzy logic is isomorphic to the algebra of functions from $[0,1]^n$ to $[0,1]$, generated by projections π_i and pointwise operations. As a consequence, for every formula φ depending on n propositional variables, we can consider its corresponding function $f_\varphi : [0,1]^n \to [0,1]$, whose value on each point is exactly the evaluation of the formula with respect to the same variable assignment. In particular, the zero and one formulas $\bar{0}, \bar{1}$ correspond to the constant functions equal to 0 and 1 respectively. Further, given two formulas φ, ψ and $\circ \in \{\wedge, \vee, \otimes, \oplus, \Rightarrow, \rightarrow\}$, for every $(x_1, \ldots, x_n) \in [0,1]^n$:

$$\begin{aligned}
&f_{\bar{0}}(x_1, \ldots, x_n) = 0 \\
&f_{\bar{1}}(x_1, \ldots, x_n) = 1 \\
&\pi_i(x_1, \ldots, x_i, \ldots, x_n) = x_i \\
&f_{\bar{\neg}\varphi}(x_1, \ldots, x_n) = \neg f_\varphi(x_1, \ldots, x_n) \\
&f_{\varphi \bar{\circ} \psi}(x_1, \ldots, x_n) = f_\varphi(x_1, \ldots, x_n) \circ f_\psi(x_1, \ldots, x_n)
\end{aligned} \tag{2.3}$$

where $\bar{\neg}, \bar{\circ}$ denote the syntactic logical connectives in formulas corresponding to the truth functions according to Table 2.2. For short, in the following we will denote at the same way both the connective and the corresponding operation if there is no reason of meaning confusion.

In the remaining of this section, we recall the main results for the fundamental fuzzy logics and we recommend [3] for the proofs and some technical details in definitions.

Gödel Logic:

Definition 2.2. *A function $f : [0,1]^n \to [0,1]$ is said an n–ary Gödel function if for every G_n–region C_ρ there exists a function $g \in \{f_{\bar{0}}, \pi_1, \dots, \pi_n, f_{\bar{1}}\}$, with*

$$f(t) = g(t)$$

for each $t = (t_1, \dots, t_n) \in C_\rho$.

Theorem 2.2. *The class of $[0,1]$–valued functions defined on $[0,1]^n$ that correspond to formulas of propositional Gödel logic coincides with the class of Gödel functions defined on $[0,1]^n$ and equipped with pointwise defined operations.*

Łukasiewicz Logic:

The fundamental result about the functional representation of **Ł**–formulas is given by the well-known McNaughton Theorem [3,114]. It states that, for the propositional case, the algebra of formulas of Łukasiewicz Logic on n variables is isomorphic to the algebra of McNaughton functions defined on $[0,1]^n$.

Definition 2.3. *Let $f : [0,1]^n \to [0,1]$ be a continuous function, f is said a McNaughton function if it is piecewise linear with integer coefficients, that is, there exists a finite set of linear functions $p_1, \dots, p_m$ with integer coefficients such that for all $(x_1, \dots, x_n) \in [0,1]^n$, there exists $i \leq m$ such that*

$$f(x_1, \dots, x_n) = p_i(x_1, \dots, x_n) \ .$$

Theorem 2.3 (McNaughton Theorem)**.** *The class of $[0,1]$–valued functions defined on $[0,1]^n$ that correspond to formulas of propositional Łukasiewicz logic coincides with the class of McNaughton functions defined on $[0,1]^n$ and equipped with pointwise defined operations.*

Product Logic:

Definition 2.4. *A function $f : (0,1]^n \to (0,1]$ is said a monomial function of order n if there exist $m_i \in \mathbb{Z}$ such that for all $(x_1, \ldots, x_n) \in [0,1]^n$*

$$f(x_1, \ldots, x_n) = 1 \wedge \prod_{i=1}^{n} x_i^{m_i} \; .$$

A function $f : (0,1]^n \to [0,1]$ is said a piecewise monomial function if exists a finite family $\{f_{pq}\}$ of monomial functions with

$$f = \bigwedge_p \bigvee_q f_{pq} \; .$$

Theorem 2.4. *The class of $[0,1]$–valued functions defined on $[0,1]^n$ that correspond to formulas of propositional Product logic coincides with the class of functions $f : [0,1]^n \to [0,1]$ such that for every $\epsilon \in \{1,2\}^n$ with $\#\epsilon = k$, the restriction of f to G_ϵ either is equal to 0 or is a k-variable piecewise monomial function.*

2.3 Fuzzy Aggregation Functions

In the previous section, we introduced some properties of continuous t-norm fuzzy logics, where a t-norm is a binary function generalizing the boolean AND to the unit interval $[0,1]$. This can be thought of as a possible way to aggregate two input fuzzy values into a single output fuzzy value, but we may be interested in aggregating more fuzzy values at once. For instance, in decision making, we are given a set of values typically representing preferences or satisfaction degrees restricted to the unit interval to be aggregated. However, once some values in the unit interval $[0,1]$ are given, there are several ways to aggregate them into a single value expressing an overall combined score, according to what is expected from such mappings. Both t-norms and t-conorms are prominent examples of *aggregation functions*, in particular they are also called *conjunctive* and *disjunctive* aggregation functions respectively. The purpose of aggregation functions is to combine inputs that are typically interpreted as degrees of membership in fuzzy sets, degrees of preference or strength of evidence. Aggregation functions have been studied by several authors in the literature [10, 15, 115], and they are successfully used in many practical applications, for instance see [60, 118, 143].

Basic Definitions

Aggregation functions are defined for inputs of any cardinality, however for simple we will give the main definitions for the case of binary aggregation functions.

Definition 2.5. *A (binary) aggregation function is a nondecreasing function* $A : [0,1]^2 \to [0,1]$*, such that:*

$$A(0,0) = 0, \qquad A(1,1) = 1 \ .$$

In particular they can be divided into four classes, i.e. *conjunctive, disjunctive, averaging* and *hybrid*, according to the pointwise order as already defined in equation (2.1) for the t-norm case,

$$A_1 \leq A_2 \quad \text{iff} \quad A_1(x,y) \leq A_2(x,y), \ \text{for all } x,y \in [0,1] \ . \tag{2.4}$$

Definition 2.6. *An aggregation function A is said to be:*

- conjunctive *when*
$$A \leq \min \ ,$$
- disjunctive *when*
$$\max \leq A \ ,$$
- averaging *(a* mean*) when*
$$\min \lneq A \lneq \max \ ,$$
- hybrid *(*mixed*) otherwise.*

Conjunctive type functions, combine values as if they were related by a logical AND operation, e.g. t-norms. On the other hand, disjunctive type functions combine values as an OR operation, e.g. t-conorms. Averaging type functions are located between minimum and maximum, which are the bounds of the t-norms and t-conorms (the Gödel t-norm and t-conorm respectively). Differently from conjunctive and disjunctive functions, that are quite sensitive to small and big inputs respectively, averaging type functions have the property that low values of some criteria can be compensated by high values of the other criteria functions.

Definition 2.7. *In the following are reported the main properties for an aggregation function A. In particular A is said to be:*

- commutative *when*

$$A(x,y) = A(y,x) \quad \textit{for all } x, y \in [0,1] \ ,$$

- associative *when*

$$A(A(x,y),z) = A(x,A(y,z)) \quad \textit{for all } x, y, z \in [0,1] \ ,$$

- idempotent *when*

$$A(x,x) = x \quad \textit{for all } x \in [0,1] \ ,$$

- *has a* neutral element *when exist* $e \in [0,1]$ *such that*

$$A(x,e) = A(e,x) = x \quad \textit{for all } x, \in [0,1] \ .$$

It is worth noticing that averaging functions (means) correspond to idempotent aggregation functions (excluding the $\min, \max$ aggregations) because of monotonicity of aggregation functions. Averaging is the most common way to combine inputs, since it assumed the total score cannot be above or below any of the inputs, but is seen as a sort of representative value of all the inputs. In fact, it is commonly adopted in voting, decision making, statistical analysis, and so on. In the following, we report some remarkable examples, while for more details on averaging functions, we recommend e.g. [10, 14].

Example 2.2.

$$\textit{(Arithmetic mean)} \qquad M_n(x_1,\dots,x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.5}$$

$$\textit{(Geometric mean)} \qquad G_n(x_1,\dots,x_n) = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} \tag{2.6}$$

$$\textit{(Harmonic mean)} \qquad H_n(x_1,\dots,x_n) = n\left(\sum_{i=1}^{n} \frac{1}{x_i}\right)^{-1} \tag{2.7}$$

We notice that $H_n \leq G_n \leq M_n$ for every $n \in \mathbb{N}$.

Despite averaging functions have nice properties to aggregate fuzzy values, they are not suitable to represent neither a conjunction nor a disjunction, indeed they not even generalize their boolean counterpart. This is a reason why, in the next section, we focus on t-norms and t-conorms, that are associative, commutative aggregation functions with 1 and 0 as neutral element, respectively. However, it is worth to mention the lot of study in defining logical connectives from general aggregation functions. For instance in [24, 89, 99], many authors have considered the use of alternatives for fuzzy conjunctions and disjunctions with respect to t-norms and t-conorms. This also allows us to define other fuzzy connectives, in particular the notion of implication related to more general aggregation functions has been the subject of many investigations [2,100,115,149]. However a deeper analysis of these approaches is beyond the scope of this work.

2.3.1 Generated Archimedean T-Norms

In mathematics, t-norms (or triangular norms in full) [78, 81] are a special kind of binary operations on the real unit interval $[0, 1]$ especially used in engineering applications of fuzzy logic. In Example 2.1 we presented some cases of interest, however in the literature a wide class of t-norms has been considered. In addition, there are several techniques to construct customized t-norms that are more suitable to deal with a certain problem, e.g. by rotations, ordinal sums of already known t-norms or defining a parametric class. In this section, we describe the case of *archimedean* t-norms [80], a special class of t-norms that can be constructed by means of unary monotone functions, called *generators*.

Definition 2.8. *A t-norm T is said to be* archimedean *if for every $x \in (0,1)$, $T(x,x) < x$. In addition, an archimedean t-norm is said* strict *if for all $x \in (0,1)$, $0 < T(x,x) < x$ otherwise is said* nilpotent.

For instance, indicating by $T_M, T_Ł$, and T_P the Gödel, Łukasiewicz and Product t-norms respectively, we have the following:

- $T_Ł$ is nilpotent and T_P is strict;
- T_M is not archimedean, indeed $T_M(x,x) = x$, for all $x \in [0,1]$.

Moreover, as reported e.g. in ([81], Propositions 5.9, 5.10), Łukasiewicz and Product t-norms are enough to represent the whole classes of nilpotent and strict archimedean t-norms.

Theorem 2.5. *Any strict t-norm is isomorphic to T_P and any nilpotent t-norm is isomorphic to $T_Ł$.*

In particular, we are interested in such archimedean t-norms that are expressible by means of a unary function, called *generator*. In order to allow using non-bijective generators, which do not have the inverse function, we employ the following notion of pseudo-inverse function:

Definition 2.9. *Let g be a monotone function, such that $g : [a,b] \to [c,d]$ where $[a,b], [c,d] \subseteq \mathbb{R} \cup \{\pm\infty\}$ are two closed subintervals of the extended real line. The* pseudo-inverse *function to g is the function $g^{(-1)} : [c,d] \to [a,b]$, defined as:*

$$g^{(-1)}(y) = \begin{cases} \sup\{x \in [a,b] : g(x) < y\} & \text{if } g(a) < g(b) \\ \sup\{x \in [a,b] : g(x) > y\} & \text{if } g(a) > g(b) \\ a & \text{if } g(a) = g(b) \end{cases}$$

A fundamental result for the construction of t-norms by additive generators is based on the following theorem (e.g. [79], Theorem 2.3):

Theorem 2.6. *Let $g : [0,1] \to [0,+\infty]$ be a strictly decreasing function with $g(1) = 0$ and $g(x) + g(y) \in Range(g) \cup [g(0^+), +\infty]$ for all x, y in $[0,1]$. Then the function $T : [0,1] \to [0,1]$ defined as*

$$T(x,y) = g^{(-1)}\left(g(x) + g(y)\right)$$

is a t-norm and g is said an additive generator *for T.*

Further, given a certain generator of a t-norm, we can define a class of related t-norms depending on a certain parameter whose extreme cases tend to be well-known t-norms.

Lemma 2.1. *Given an additive generator function g of a t-norm T and $\lambda > 0$, then T^λ, corresponding to the generator function $g^\lambda(x) = (g(x))^\lambda$ denotes a class of increasing t-norms and we have:*

$$lim_{\lambda \to 0^+} T^\lambda = T_D \qquad and \qquad lim_{\lambda \to \infty} T^\lambda = T_M \ ,$$

where T_D and T_M denote the drastic *and min t-norms respectively.*

As shown in [79], any t-norm T with an additive generator g is archimedean, further if g is continuous then T is continuous, archimedean and T is strict if and only if $g(0) = +\infty$, otherwise T is nilpotent. In order to avoid using the notion of pseudo-inverse function one may exploit the equivalent expression

$$T(x,y) = g^{-1}\left(\min\{g(0^+), g(x) + g(y)\}\right) \ . \tag{2.8}$$

Example 2.3. *If we take $g(x) = 1 - x$ as additive generator, then also $g^{-1}(y) = 1 - y$ and we get the well-known Łukasiewicz t-norm $T_Ł$:*

$$T(x,y) = 1 - \min\{1, 1 - x + 1 - y\} = \max\{0, x + y - 1\} \ .$$

Example 2.4. *If we take $g(x) = -log(x)$ as additive generator, then we have $g^{-1}(y) = e^{-y}$ and we get the well-known Product t-norm T_P:*

$$T(x,y) = e^{-(\min\{+\infty, -log(x) - log(y)\})} = x \cdot y \ .$$

The isomorphism between addition on $[0,+\infty]$ and multiplication on $[0,1]$ by the logarithm and the exponential functions allows two-way transformations between additive and multiplicative generators of a t-norm. If g is an additive generator of a t-norm T, then the strictly increasing function $h : [0,1] \to [0,1]$ defined as $h(x) = e^{-g(x)}$ is a multiplicative generator of T, namely:

$$T(x, y) = h^{-1}(\max(h(0), h(x) \cdot h(y)))$$

On the opposite, if h is a multiplicative generator of T, then $g(x) = -log(h(x))$ is an additive generator of T. For instance, $h(x) = e^{x-1}$ and $h(x) = x$ are multiplicative generators of $T_{Ł}$ and T_P, respectively. Additive and multiplicative generators are isomorphic and we decide to focus in the former for simplicity. We only mention that both multiples of additive generators and positive powers of multiplicative generators determine the same t-norm.

An interesting consequence of equation (2.8) is that it allows us to define also the other fuzzy connectives, deriving from the t-norm, as depending on the generator. In particular, we have:

$$\text{residuum :} \quad x \Rightarrow y = g^{-1}\left(\max\{0, g(y) - g(x)\}\right) \tag{2.9}$$

$$\text{bi-residuum :} \quad x \Leftrightarrow y = g^{-1}\left(|g(x) - g(y)|\right) \tag{2.10}$$

These representations will be exploited in the next chapters to define opportune loss functions from FOL logical constraints. In particular, they also allow us to analyze the representation of more complex truth functions with multiple occurring connectives.

Parametrized classes of t-norms

Constructing t-norms by generators may be extended to the construction of classes of t-norms determined by a generator function depending on a certain parameter. Several parametrized families of t-norms have been introduced and studied in the literature. In the following we recall some prominent examples.

Example 2.5 (Schweizer-Sklar t-norms). *Consider the following parametrized generator for $\lambda \in (-\infty, +\infty)$:*

$$g^{SS}_{\lambda}(x) = \begin{cases} -log(x) & \text{if } \lambda = 0 \\ \frac{1-x^{\lambda}}{\lambda} & \text{otherwise,} \end{cases}$$

The class of t-norms corresponding to this generator are called Schweizer-Sklar t-norms, and it is defined as:

$$T_\lambda^{SS}(x,y) = \begin{cases} T_M(x,y) & \textit{if } \lambda = -\infty \\ (x^\lambda + y^\lambda - 1)^{\frac{1}{\lambda}} & \textit{if } -\infty < \lambda < 0 \\ T_P(x,y) & \textit{if } \lambda = 0 \\ \max(0, x^\lambda + y^\lambda - 1)^{\frac{1}{\lambda}} & \textit{if } 0 < \lambda < +\infty \\ T_D(x,y) & \textit{if } \lambda = +\infty \end{cases}$$

A Schweizer-Sklar t-norm T_λ^{SS} is archimedean *if and only if $\lambda > -\infty$,* continuous *if and only if $\lambda < +\infty$,* strict *if and only if $-\infty < \lambda \leq 0$ and* nilpotent *if and only if $0 < \lambda < +\infty$. This t-norm family is strictly decreasing for $\lambda \geq 0$ and continuous with respect to $\lambda \in [-\infty, +\infty]$.*

Example 2.6 (Hamacher t-norms)**.** *Consider the following parametrized generator for $\lambda \in [0, +\infty)$:*

$$g_\lambda^H(x) = \begin{cases} \frac{1-x}{x} & \textit{if } \lambda = 0 \\ log(\frac{\lambda + (1-\lambda)x}{x}) & \textit{otherwise.} \end{cases}$$

The class of t-norms corresponding to this generator are called Hamacher t-norms, and they are the only t-norms which are rational functions.

$$T_\lambda^H(x,y) = \begin{cases} 0 & \textit{if } \lambda = x = y = 0 \\ \frac{xy}{\lambda + (1-\lambda)(x+y-xy)} & \textit{otherwise.} \end{cases}$$

A Hamacher t-norm T_λ^H is strict *if and only if $\lambda < +\infty$ (for $\lambda = 1$ we get the Product t-norm). This t-norm family is strictly decreasing and continuous with respect to any $\lambda \in [0, +\infty]$.*

Example 2.7 (Yager t-norms)**.** *Consider the following parametrized generator for $\lambda \in (0, +\infty)$:*

$$g_\lambda^Y(x) = (1-x)^\lambda$$

The class of t-norms corresponding to this generator are called Yager t-norms, *and are defined as:*

$$T^Y_\lambda(x,y) = \begin{cases} T_D(x,y) & \textit{if } \lambda = 0 \\ \max\{0, 1 - \big((1-x)^\lambda + (1-y)^\lambda\big)^{\frac{1}{\lambda}}\big)\} & \textit{if } 0 < \lambda < +\infty \\ T_M(x,y) & \textit{if } \lambda = +\infty \end{cases}$$

A Yager t-norm T^H_λ *is* nilpotent *if and only if* $0 < \lambda < +\infty$ *(for* $\lambda = 1$ *we get the Łukasiewicz t-norm). This t-norm family is strictly increasing and continuous with respect to any* $\lambda \in [0, +\infty]$. *We also note that the Yager t-norm arises from the Łukasiewicz t-norm by raising its additive generator to the power of* λ *as described in Lemma 2.1.*

2.4 Machine Learning

Learning can be naively described as the process of gaining some knowledge by experience and modification of a behavioral tendency. As regards *machine learning* (ML), we might say that a machine learns whenever it changes its parameters in order that its expected future performance improves. The performance is generally measured on a certain task and different performance measures can be applied to different tasks. In particular, ML deals with tasks associated to artificial intelligence such as pattern recognition, diagnosis, planning, robot control, prediction, etc. In the following, we only introduce some basic techniques of machine learning and we recommend, for instance [57,122] for a more comprehensive coverage of the fundamentals and [55] whose main focus is on *Deep Learning*, a successful subfield of ML where deep architectures are exploited in order to learn very complex functions.

The field of Machine Learning [112,122] defines theories and a wide range of techniques, which can be used to approximate an unknown function. A classical starting point for most ML approaches is to define a cost (loss) functional (the same reasoning can be applied to probabilistic methods, where the cost functional is replaced by a likelihood). The cost functional depends on the parameters of the approximators and it assumes lower values for functions having a closer-to-desired behavior. For example, the cost functional can express how the function should behave on some data points or it can penalize values outside a given range or force the function to respond similarly to different pairs of inputs. Once the cost functional is defined, the training of the learner becomes an optimization problem with respect to the parameters, which can be tackled via gradient descent or other optimization techniques. This is the reason why, one desired property of a cost functional is to be convex and in Chap. 4 we discuss the case of a convex optimization problem.

2.4.1 Tasks

High-level human capabilities, such as speech-recognition, language understanding and so on, might be too difficult to be replicated with fixed programs. This is the reason why machine learning approach consists in describing how the learning system has to process data and consequently modify its parameters. In this view, learning can be considered as the way we attain the ability to perform the task. A non-exhaustive list of some of the most common machine learning tasks is reported in the following.

classification. In a *classification* task we are interested in learning a map f that assigns a category $f(x) \in \{1, \dots, k\}$ to any input $x \in \mathbb{R}^n$, namely we are asked to specify which of a limited set of categories an input belongs to. As an example, consider the case in which the input is given by an image x represented as a vector whose components are the pixel values of the image and the task is to predict if x contains a human face. In this case, the task can be defined as the problem of learning a $\{0, 1\}$–valued function f such that, $f(x) = 0$ means that the image x does not contain a human face, whereas $f(x) = 1$ means that it does.

regression. This task is similar to classification except the learning algorithm is asked to output a function $f : \mathbb{R}^n \to \mathbb{R}$. For instance, consider the case of predicting house prices given its square footage, and so on.

anomaly detection. Consider the task of credit card fraud detection. In this case, the goal is to predict a fraud given an unusual behavior with respect to some purchase habits. This is an example of a wide range of techniques aiming to discover outliers from a certain probability distribution, that is also referred as *anomaly detection*.

synthesis and sampling. Machine learning algorithms can also be exploited to generate new examples that are similar to those available in the training data, this is the case of *synthesis and sampling.* For instance, if only a small portion of data is known, we could employ these techniques to increase the number of available examples according to the data distribution or, as in the case of Generative Adversarial Networks (GANs), to generate new patterns seeming realistic as the original ones.

missing features. Let us consider the case we are given a set of data where the inputs are only partially known. In a *missing features* task, the goal is to determine the unknown components x_is of a certain input $x \in \mathbb{R}^n$. As an example, consider the case we are given a new point $x = (x_1, x_2) \in \mathbb{R}^2$ and we only know its abscissa. For instance, we could infer its ordinate according to the whole dataset distribution. In we know x has to be classified in a certain category k, we should determine x_2 with respect to the membership degree of x to k.

density estimation. Sometimes can be useful to consider learning tasks where the unknown function is a probability density (or probability mass if x is discrete) function $\rho : \mathbb{R}^n \to \mathbb{R}$, in this case we talk about *density estimation.*

clustering. Given a set of objects, the task of grouping them in such a way that objects in the same group (called a cluster) are more similar (according to a certain criterion) to each other than to those in other groups is said *clustering.* In general, the term clustering does not refer to one specific algorithm, but to the general task to be solved. Indeed, the objects may be grouped according to a certain measure distance or a statistical distribution. Clustering techniques have been successfully applied into several fields, as for instance in computational biology, marketing, social science, and so on.

Performance Measure As we pointed out, learning can be seen as the process of becoming capable of doing a certain task and this ability has to be quantitatively evaluated by opportune *performance measures.* Since machine learning may carry out very different tasks, it is fair to argue that different kinds of evaluations have to be applied and any task may be subject to one or more metrics with various meanings. For instance, a common choice in task like classification is the *accuracy*, that simply is the proportion of examples for which the unknown function produces the correct output. In particular, we are interested in measuring the accuracy of the system on data that is unseen during the training phase, also said *test* data, this allows us to evaluate the actual performance in real world applications.

2.4.2 Types of Algorithms

One of the main reason for the recent success of machine learning techniques, in particular deep learning, is the availability of large amounts of (labeled) data. Any example in a dataset is generally represented as a real-valued vector $x \in \mathbb{R}^n$, where any component x_i denotes the value corresponding to a certain feature of the example. For instance, if the domain consists of a set of people, any person could be represented as a vector containing some features, e.g. *age, height, weight* and so on. It is worth noticing that the involved features should be relevant for the considered task, otherwise the learning problem could be ill-posed or too difficult to be solved. Roughly speaking, if the task is, for instance, the classification of the prosperity of a person, it may be misleading to make such a prediction based on its age, height or weight. However in general, it is not trivial to establish which are the "right" features to be considered for a certain task. *Labels*, also said *targets*, can be attached to the training examples in order to provide the desired behavior for the unknown function to be learned. In a classification problem labels are generally the values $0, 1$ or ± 1. If for a certain example x we are given its label y, the pair (x, y) is said a *supervised example*, otherwise x is said *unsupervised*. In case of multi-task learning problems, namely if we have to learn more than a single function, we talk about *fully labeled* problem if for any example we are given the labels of any learnable function, otherwise is said *partially labeled*. According to the availability of targets for the examples in a certain learning problem, we can consider different learning paradigms.

Unsupervised Learning Let us consider a learning problem where no example is provided with a target. In this case, we talk about *unsupervised* *learning* and the goal is generally the identification of common structures, patterns or features among the examples to improve the analysis of new data. A typical unsupervised learning task is *clustering* that consists in grouping a set of objects, according to a certain similarity criterion, into different groups said clusters. However in general, similarity is expressed by a distance function that can be difficult to be chosen if the input space has a high dimensionality. Some common unsupervised learning algorithms are: *k-means* (clustering), *local outlier factor* (anomaly detection), *autoencoders* (neural networks).

Supervised Learning If for a certain example x in input, the desired output y for a function f to be learned is available, we can enforce the prediction $f(x)$ to be as close as possible to y. *Supervised learning* assumes a dataset consisting of examples provided with targets and the goal is to learn one or more unknown functions according to these input-output pairs. In particular, the functions are learned exploiting a subset of the whole available dataset said *training set*, and the capacity to generalize to new data is measured on another portion of data said *test set*. Some common machine learning methods are widely used in supervised learning algorithms, for instance: *Support Vector Machines* (SVMs), *Naive Bayes*, *Artificial Neural Networks* (ANNs).

Other Learning Approaches Supervised and unsupervised learning algorithms do not provide a strict categorization. Indeed, other variants of the learning paradigm are possible. For instance, consider the case of a partially labeled problem or, more in general, a case in which only a subset of the available data is provided with target. This is a common setting in real world applications where only a small portion of data is generally labeled, but we are given a (possibly) huge amount of unsupervised examples. Such problems are referred to as *semi–supervised learning*.

As it is clear, there are several possible ways to deal with a dataset. Another interesting example is the case of *reinforcement learning*, where the learning algorithm interacts with an environment by means of a feedback loop. In particular, for any action carried out by the learning system, the environment provides a reward. The main idea is that actions associated with greater rewards will be encouraged (reinforced), whereas the others will be weakened.

Regularization

A special role in designing efficient machine learning algorithms is played by *regularization*. In general, the term regularization refers to a set of modifications we make to a learning algorithm to express some preference among its possible solutions. For instance, consider the case we have to learn an affine function and different solutions are eligible. We may express the preference for solutions that have a smaller slope adding a penalty term called regularizer in the optimization objective. Smoother functions are assumed to generalize better, indeed they return more similar predictions for similar inputs than irregular solutions. If such preferences are aligned with the problem we want to solve, the algorithm is shown to get better performance. Please note that, even if regularization can be exploited to reduce the generalization error, in general it may increase the training error.

From a philosophical point of view, regularization can be thought of as an application of the well-known *principle of parsimony* (Occam's razor). This principle states that among equally performing different explanations for the known observations, we should choose the "simplest" one. Regularization can also be seen from a probabilistic point of view. Indeed these techniques are equivalent to the imposition of certain prior distributions on model parameters. In the end, there are many ways to regularize a certain learning problem. However, there is not a best form of regularization (as shown by the *No Free Lunch Theorem* [152]) and we must choose a form that is well suited to the particular task we want to solve.

2.4.3 Some Models

In the previous pages, we introduced some basic definitions for machine learning as well as some nomenclature that will be used in the following chapters. In the remaining of this section, we describe the main aspects of some ML models without pretending to be exhaustive. We will refer to appropriate references for more details.

Support Vector Machines

Support vector machines (SVMs) are a class of kernel methods originally conceived by Vapnik and Chervonenkis [20]. One of the main advantages of this approach is the capacity to create nonlinear classifiers by applying the kernel trick to maximum-margin hyperplanes [12, 20]. This property derives from the implicit definition of a (possibly infinite) high-dimensional feature representation of data determined by the chosen kernel. Actually, the class of methods exploiting the kernel trick is wider than SVMs only and is known as *kernel machines* [134, 150]. One general drawback of kernel machines is that the cost of evaluating the decision function is linear in the number of training examples. However, support vector machines overcome this limitation, indeed it turns out that only a portion of the training data is significant to determine the maximum-margin separating hyperplane in the feature space, the so called *support vectors*. In the following we sketch the main steps of the classic supervised SVMs model.

Given a set of supervised examples $\mathcal{X} = \{(x_l, y_l) : \ l = 1, \ldots, L\}$ for a task function f, the learning strategy consists in the minimization of a regularization term $||\omega||^2$ subject to a set of constraints $y_l \cdot f(x_l) \geq 1$ that enforce the membership of the example points $x_l \in \mathbb{R}^n$ to the positive (+1) or negative (-1) class, as specified by the corresponding targets y_l. In SVMs the function f is assumed to be belong to a certain *Reproducing Kernel Hilbert Space* (RKHS) [5] with kernel function k, hence f can be expressed as an expansion of k. Given the kernel properties, f can also be expressed as an affine function of a high-dimensional feature representation ϕ of the input determined by k, namely $f(x) = \omega \cdot \phi(x) + b$, where $k(x, y) = \langle \phi(x), \phi(y) \rangle$. The optimization problem can be defined as

$$\min_{\omega} \frac{1}{2} ||\omega||^2$$

`subject to:`

$$y_l \cdot f(x_l) \geq 1, \ \text{for every } l = 1, \ldots, L$$

The satisfaction of the constraints can also be obtained by the minimization of a hinge loss function not penalizing output values “beyond” the target.

$$\min_{\omega} \frac{1}{2}||\omega||^2 + \sum_{l=1}^{L} \max\{0, 1 - y_l \cdot f(x_l)\}$$

As a consequence, the solution of the optimization problem will depend only on a subset of the given training data, namely those that contribute to the definition of the maximum-margin hyperplane separating the two classes in the feature space, the support vectors. Such points correspond to not null coefficients of the kernel expansion for the best solution of the above optimization problem and from a constrained optimization point of view, they correspond to the *active* constraints of a Lagrangian formulation. Hence, we can split the training examples into two categories, the support vectors, that completely determine the optimal solution of the problem and the straw vectors. In Chap. 5 we will extend this paradigm to a class of semi–supervised learning problems where also logical constraints are enforced on the available samples.

Artificial Neural Networks

Nowadays, one of the main successfully employed frameworks for dealing with different machine learning algorithms and process complex data inputs is given by *Artificial Neural Networks* (ANNs) [67]. This name is related to the fact ANNs take inspiration from the biological neural networks that constitute animal brains. An artificial neural network consists in a group of nodes (neurons) interconnected by weighted edges (synapses) forming a directed, weighted graph. Mathematically, every node has associated an activation function while any edge weight between two nodes, lets say from A to B, corresponds to a real number multiplying the output of A to be given as input to B. Typically, neurons are aggregated into layers and activation functions are computed by non-linear functions of the weighed sum of its inputs. Data are processed from the input layer to the output layer eventually moving trough several hidden layers. In this setting, learning can be regarded as the process of modifying the edge weights in order for the network to become capable of performing a certain task.

The concept of ANN is quite old and has evolved across the years [103,128], however they currently reach state-of-the-art performances on a variety of tasks, like computer vision, speech recognition, machine translation, medical diagnosis and so on. One of the strong point of ANNs is the capacity they have to automatically identify characteristics from the learning material they process. According to what they are modeling, ANNs architectures may be better suited to deal on certain kinds of tasks and data. In facts, a wide class of architectures have been considered in the literature and new designs to improve the performance on increasingly complex tasks are continuously under investigation. In particular it is worth to mention the well-known:

- A *Multilayer Perceptron* (MLP) is a class of feedforward artificial neural network with one or more hidden layers. MLPs allows us to approximate extremely complex functions, further they are universal approximators (see Cybenko's Theorem [23]).
- A *Convolutional Neural Network* (CNN) is a class of deep neural networks, where convolutional layers apply a convolution operation to the input, passing the result to the next layer. CNNs [86] are especially suited to deal with image classification problem, indeed the convolution emulates the response of an individual neuron to visual stimuli.
- A *Recurrent Neural Network* (RNN) is a class of artificial neural network where connections between nodes form a directed graph along a sequence. Unlike MLPs, RNNs [140] can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as language modeling, speech recognition, time series prediction, and so on.
- A *Graph Neural Network* (GNN) is a class of artificial neural network taking as input a graph, namely a set of nodes with edges among them. GNNs [133] are especially suitable to deal with structured inputs, where the connections among patterns is knowns and can be exploited by the network.

All these architectures generally depend on a set of parameters to be determined in order to learn how to perform a certain task. Even if some differences among the algorithms for training such networks have to be considered, the general adopted method is simply *Backpropagation* [130] (backward propagation of errors). Backpropagation is a method to distribute an error computed at the output layer of an ANN throughout the network's layers. In particular, it is commonly used by the gradient descent optimization algorithm to adjust the weight of neurons by calculating the gradient of the loss function.

2.4.4 ML as Learning from Constraints

In this section, we introduced basic definitions of machine learning theory presenting some of the main arguments that are involved in this thesis. As we already noticed, ML includes a wide range of techniques and algorithms, however as claimed e.g. in [57], the whole theory can be framed around the unifying notion of *constraint*. The concept of constraint has been considered in the definition of general approaches for learning that allow us to exploit different sources of knowledge for training an agent [57, 129]. While the constraints formally represent the available information about the environment, the agent is supposed to learn specific tasks satisfying them [44, 54].

Learning from constraints is an extension of classical supervised learning in which the concept of *constraint* is used to refer to a more general class of knowledge than simple labeled examples. An extensive investigation of different kinds of constraints can be found in [53], where variational calculus is exploited to generalize the Representation Theorem for kernel machines. In general, whenever we are dealing with a multi-task learning problem in which the functions to be learned are subject to a set of relational bonds, we can exploit constraints to formalize these interdependency. This is exactly the adopted paradigm of *constraint programming* (CP) [101], initially investigating only logic programming languages [68], where different constraints are embedded into a host language. In CP, different relations among variables are stated by various kinds of constraints like logical formulas, linear inequalities, etc. and the main goal is to determine the assignation for the unknown variables satisfying the largest number of constraints at the same time. The integration of constraint-based techniques with machine learning has been considered, and is still under investigation, by several authors in the literature [9, 31, 119], so as its effectiveness with respect to deep learning architectures [113]. As illustrated in Chap. 4, in this work we mostly focus on logical constraints, which provide an expressive and formally well-defined representation for abstract knowledge.

Chapter 3

Learning and Logical Reasoning

In real-world applications it is not uncommon to deal with uncertain and vague information to face a certain task. This is one of the main reasons why many frameworks, with slightly different representation formalisms [131], arose from the integration of relational systems and probabilistic models as particular subdisciplines of artificial intelligence. All these frameworks can be grouped under the broader field of *statistical relational learning* (SRL) [74, 117], that is equally concerned with learning, reasoning and knowledge representation. One of the most employed formalism to represent relational knowledge in SRL is given by first–order logic (FOL) syntax and we mostly focus on approaches based on this language. In this chapter, we present some related literature and some of the most studied systems dealing with learning and symbolic reasoning on logical representations of a certain knowledge base.

3.1 Related Works

There are several approaches to embed logical knowledge into learning processes. Minervini et al. [102] propose to use prior knowledge to correct the inconsistencies of an adversarial learner. Their methodology is designed ah-hoc for the tackled task, and limited to Horn clauses. Related approaches also combine logic rules with neural networks. For instance, Hu et al. [66] propose an iterative procedure that transfers abstract knowledge encoded by the logic rules into the parameters of a deep neural network. Their method is also based on a fuzzy generalization of FOL. However, the definition of the framework is limited to universally quantified formulas and to a small set of logic operators. Moreover, the connections between logic and *kernel machines* have been the subject of many investigations. For instance, in [22] a family of kernel functions is built up from a Feature Description Language to exploit the relational structure of the domain. One limitation of this work is that the integration with the logic formalism does not reveal very tight connections. A different approach is considered in [34, 137], where the t-norm theory is used to translate first–order logic formulas into real-valued functions, the logical constraints. In [33] the authors were able to extend the classical regularization scheme of kernel machines to incorporate logical constraints. Unfortunately, the objective function depending on the logical constraints turns out to be not convex in general, unless one restricts the attention to only Horn clauses [35]. With respect to this point, the fragment we present in Sec. 4.1 provides the complete set of formulas that can be written in Łukasiewicz logic to get convex functional constraints. In particular, we are interested in methods exploiting the transcription of logical rules into real valued functions to define appropriate cost functions to be optimized in the learning process. Compared to this, fuzzy logic (see Sec. 2.2) turns out to be a valuable choice in defining the mapping between formulas and constraints since it provides continuous operators as truth functions evaluated in the real-unit interval.

In the following, we present an overview of some prominent frameworks that are related to the integration of symbolic and sub-symbolic processes for learning and inference.

ILP. One of the best known research area that combines logic programming with machine learning techniques is *Inductive Logic Programming* (ILP) [107, 109]. The general inductive problem is as follows: given a set of positive P and negative N examples and a consistent background knowledge B, find a hypothesis H such that the conjunction of H and B entails all the examples of P and none of N. A large number of hypotheses typically fits such a definition. For instance the Bayesian ILP setting [108] assumes a prior probability distribution defined over the hypothesis space. In [27] clauses are given a probability value and two methods to estimate these parameters and the hypothesis are provided. In addition, it is worth to mention some related works on Inductive Logic Programming and kernel machines like [84] and [110]. In the first paper the learning algorithm *first–order inductive learner* (FOIL) is combined with kernel methods by leveraging FOIL search for a set of relevant clauses. The latter paper defines a kernel, that is an inner product in the feature space spanned by a given set of first-order hypothesized clauses.

MLN and PSL. As we already mentioned, statistical relational learning [49] deserves a special mention in this overview. SRL focuses on relational domains under uncertainty, where relations among objects are expressed by first–order logic formulas and uncertainty is handled by setting up probabilistic graphical models, like Bayesian networks [70]. Probabilistic logic learning has been the subject of many investigations from several authors and we recommend [26] for a survey. In these years, a lot of frameworks arose into this field, among which emerged *Markov Logic Networks* (MLNs) [120] and *Probabilistic Soft Logic* (PSL) [13, 75]. Formally an MLN is a set of weighted FOL formulas, but can be viewed as a template for constructing Markov Random Fields (MRFs) [77] to model the joint distribution of the set of all the possible atomic groundings in formulas. Nevertheless, in such approach logical formulas are assumed to be evaluated as true-false values, whereas a more general expressiveness is achieved in PSL where formulas can take any $[0, 1]$ value. Analogously to MLN, PSL can be viewed as a template for Hinge–Loss Markov Random Fields, that are continuous generalization of MRFs whose formulas are restricted to disjunctive clauses (including Horn clauses as a special case) and translated by the Łukasiewicz t-norm and t-conorm. With this restriction, the *most probable assignment* (MAP) to the unknown variables reduces to a convex optimization problem [8] avoiding the general intractability of MLNs and they also provide different approaches to estimate the rule weights [6]. As we point out in Sec. 5.4, where a more accurate comparison with PSL is discussed, the set of formulas keeping convexity in this frame can be extended.

ProbLog and DeepProbLog. Symbolic reasoning [29, 73, 75], which is typically based on logic and probability, allows us to perform high-level inference (possibly under uncertainty) without having to deal with thousands of learning hyper-parameters. Over the years, several *probabilistic logics* [42] e.g. PRISM [132] and *Stochastic Logic Program* (SLP) [111], have been introduced with the aim of carrying out both learning and reasoning task, exploiting logical representation and probabilistic inference. For instance *ProbLog* [28], a probabilistic extension of *Prolog* [18], builds programs encoding interactions for a large sets of facts (clauses), each associated with a certain uncertainty. In particular, a ProbLog program T specifies a probability distribution over logic programs (assumed as mutually independent) by specifying for each clause the probability that it belongs to a certain sub-program of T. Then, the success probability of any query is defined as the probability that it succeeds in these subprograms. However, in order to compute the success probability of queries, the enumeration of all the possible logic programs is generally not feasible. Then ProbLog convert all the programs, queries and evidence to a weighted Boolean formula whose probability can be computed, e.g. by weighted model counting, exploiting state-of-the-art methods known from the literature. Some extensions of ProbLog have been considered to both increase its expressiveness and to exploit possible integration with sub-symbolic models. For instance, in [76] the authors consider also algebraic facts, each labeled with an element of an opportune semiring, and the semiring is used to calculate labels of possible worlds and of queries. More recently, DeepProbLog has been proposed [95], a generalized version of ProbLog to integrate logical reasoning and deep learning models. The main idea is to consider the atomic expressions as neural predicates in an existing probabilistic logic programming language, in this particular case ProbLog. However a limitation of this approach is that DeepProbLog requires the output from the neural networks to be probabilities and atomic formulas to be independent, while the sub-symbolic layer often consists of several neural layers sharing weights.

ProPPR, TensorLog and Neural LP. Even if recent work has tried to gain insight on how a deep model works [94], sub-symbolic approaches are still mostly seen as *black-boxes*, whereas symbolic approaches are generally easier to interpret, as the symbol manipulation or chain of reasoning can be unfolded to provide an understandable explanation to a human operator. An interesting extension of SLP that enables efficient learning and inference on graphs is given by ProPPR [148]. ProPPR is a probabilistic logic generating first–order theories via parameter learning, where inference can be carried out restricting on small graphs of local groundings. This property guarantees the scalability of the approach with respect to large database. For instance in [147], it is shown how to learn continuous low-dimensional embeddings for first–order logic formulas from scratch, on two knowledge base completion tasks. In particular, the training examples and inference formulas are mapped into a binary matrix, then the latent continuous representations of examples and logical formulas is learned via a low-rank approximation method of matrix factorization. In [19] is presented a probabilistic deductive database, called TensorLog, in which reasoning uses a differentiable process. TensorLog is shown to be faster than ProPPR with large numbers of training examples while ProPPR should be faster for very large database and small numbers of training examples. In TensorLog logical inference is carried out by sequences of differentiable numerical operations on matrices. However, this framework is limited to learning only the parameters of the logical rules. An extension to jointly learn the structure of the rules is proposed by *Neural Logic Programming* (Neural LP) [124] a completely differentiable system for learning models defined by sets of first–order logic rules exploiting gradient-based programming frameworks and optimization methods for the inductive logic programming task. In Neural LP, parameters and structure are simultaneously learned exploiting a neural controller system with an attention mechanism and memory to perform TensorLog's operations.

SBR and LTN. Semantic–Based Regularization [34] (SBR) is a unified framework for inference and learning that is centered around the notion of constraints and of parsimony principle. The SBR goal is to find the smoothest functions satisfying the (possibly weighted) constraints. As pointed out in [34], the given solution can be interpreted also in probabilistic terms and directly compared with MLNs. In a similar direction, some recent works by Serafini et al. [136,137] propose a framework called Logic Tensor Networks (LTN) that integrates learning based on neural networks with reasoning using first–order fuzzy logic, all implemented in TensorFlow. In particular, given some data available in the form of real-valued vectors and a set of FOL clauses, LTN converts these formulas according to a chosen t-conorm (e.g. Łukasiewicz t-conorm) and the grounding of predicates on data is defined as a generalization of the neural tensor network [139]. Then, the truth value of any formula can be determined by a neural network which first computes the grounding of the literals contained in the clause, and then combines them using the specific t-conorm. Finally, the model parameters can be determined minimizing the loss function corresponding to the satisfiability error of the formulas on the available data.

In the following, we describe with more details some of the most relevant statistical relational learning frameworks that have been already introduced in this section, such as Markov Logic Networks and Probabilistic Soft Logic. In addition, we describe Semantic–Based Regularization and Logic Tensor Network, indeed the system we proposed in Ch. 6, called LYRICS, is inspired by these two frameworks.

3.2 Markov Logic Networks

Markov logic networks (MLNs) [120] implement a probabilistic logic providing a general interface to integrate learning and probabilistic inference. In particular, first–order logic is used to define boolean Markov Random Fields (MRFs). Any logical formula (possibly with a weight) corresponds to a template for a set of potentials having a higher score for such assignments that most satisfy the formula. Inference and weight learning of the logical rules involved in a learning process are carried out in MLNs, and an algorithm to learn the set of rules from scratch if presented in [82]. MLNs incorporate logical semantics defining feature functions into probability distributions to create models that capture both the structure and the uncertainty in machine learning tasks. MLNs deal with first–order logic knowledge base, however an interesting expressiveness extension has been considered in [61], where MLNs are extended to deal with statistical universal quantifiers.

In particular, MLNs rely on the notion of Markov Random Field (MRF). An MRF is a probabilistic graphical model for the joint distribution of a set of variables and it is composed of an undirected graph expressing the variable dependencies and a set of potential functions. Each variable corresponds to a node in the graph while a potential function (i.e. a non-negative function of the state of the corresponding clique) is associated to any clique of the graph.

Definition 3.1 (MRF). *Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{X}$ be a vector of random variables and let $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_m)$ be a vector of potentials, where each poten-*

tial ϕ_j assigns a real-valued score to any configuration of the variables. Given $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_m)$ a vector of real-valued weights, a Markov Random Field is a probability distribution of the form:

$$P(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{m} \omega_j \phi_j(\boldsymbol{x})\right) \ ,$$

where $Z = \int_{\mathcal{X}} \exp\left(\sum_{j=1}^{m} \omega_j \phi_j(\boldsymbol{x}')\right) d\boldsymbol{x}'$ is known as the partition function.

The integration with logic is carried out in MLNs as follows. Each potential function ϕ_j is associated to a first–order logic formula F_j in a knowledge base KB. The KB can be seen as a set of constraints on the set of possible assignment, the fewer formulas an assignment violates, the more probable it is, while it has zero probability if it violates anyone formula. Each formula has to be considered either as hard (infinite weight) or can be weighted to penalize differently the assignments with respect to the formula satisfaction, the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not.

Definition 3.2 (MLN). *A Markov logic network L is a set of pairs (F_j, ω_j), where F_j is a FOL formula and $\omega_j \in \mathbb{R}$. Relatively to a set of constants $K = \{k_1, \ldots, k_{|K|}\}$, it defines an MRF $M_{L,K}$ as follows:*

- *$M_{L,K}$ contains one binary node for each possible grounding[1] of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*

- *$M_{L,K}$ contains one feature for each possible grounding of each formula F_j in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the ω_j associated with F_j in L.*

Hence, an MLN can be viewed as a template for constructing MRFs varying different sets of constants. Each of these MRF is said a ground MRF and the probability distribution over possible assignments $\boldsymbol{x}$ specified by the

[1]A *grounding* is an evaluation of a predicate on a certain element of a domain.

ground Markov network $M_{L,K}$ is given by

$$P(\boldsymbol{x}) = \frac{1}{Z} \exp\left(\sum_{j=1}^{m} \omega_j \phi_j(\boldsymbol{x})\right) ,$$

where $\phi_j(\boldsymbol{x})$ is the number of true groundings of F_j in $\boldsymbol{x}$.

MLNs allow formulas with conjunctions, as well as negative or infinite weights. However, the full class of Markov logic networks does not admit any known polynomial-time approximation schemata for MAP inference. That is a reason why we decide to consider a KB made of logic clauses. A clause $C_j \in \boldsymbol{C}$ is a disjunction of variable x or its negation $\neg x$, well-known as literals. In particular for every $j = 1, \dots, m$, $\phi_j(\boldsymbol{x})$ equal 1 if an assignment to the variable $\boldsymbol{x}$ satisfies C_j and equal 0 otherwise. The weights of the potentials express the probability that the clause holds according to the model. Let $I_j^+, I_j^- \subseteq \{1, \dots, n\}$ be te set of indexes of the variables x_i occurring in C_j. Then C_j can be written as:

$$\left(\bigvee_{i \in I_j^+} x_i\right) \bigvee \left(\bigvee_{i \in I_j^-} \neg x_i\right) . \tag{3.1}$$

In particular, MLNs can be exploited to find a most probable assignment to the variables, i.e. MAP inference. Given an MRF defined by clauses in $\boldsymbol{C}$, MAP inference can be defined as the following integer linear program:

$$\arg\max_{x \in \{0,1\}^n} \sum_{C_j \in \boldsymbol{C}} \omega_J \min\left\{\sum_{i \in I_j^+} x_i + \sum_{i \in I_j^-} (1 - x_i), 1\right\} . \tag{3.2}$$

While this program is generally intractable, some possible convex programming relaxations have been considered in the literature, e.g. [7,8]. In the next section, we will introduce a generalization of MRFs, called HL–MRFs, that are defined on continuous variables instead of boolean variables as for MLNs. Similar to MLNs, PSL allows to define templates for HL–MRFs.

3.3 Probabilistic Soft Logic

A possible way to make the problem in equation (3.2) feasible relies on the relaxation to continuous values for the random variables in $\boldsymbol{x}$. In particular, the formulas in the knowledge base $\boldsymbol{C}$ are converted with the Łukasiewicz logic semantics (see Table 2.2) instead of Boolean logic. Łukasiewicz logic is a continuous t-norm fuzzy logic and **Ł**-propositions in $\boldsymbol{x}$ can take truth values in $[0,1]$ instead of $\{0,1\}$. This allows us to represent also vague concepts together with their uncertainty. According to Łukasiewicz logic clauses in $\boldsymbol{C}$, the problem in equation (3.2) can be reformulated as follows.

$$\operatorname*{arg\,max}_{x\in[0,1]^n} \sum_{C_j\in\boldsymbol{C}} \omega_j \min\left\{\sum_{i\in I_j^+} x_i + \sum_{i\in I_j^-}(1-x_i), 1\right\} . \tag{3.3}$$

Whenever a clause C_j is unsatisfied we can express its distance to satisfaction. In particular the problem in equation (3.3) can be rewritten in the following convex optimization problem:

$$\operatorname*{arg\,min}_{x\in[0,1]^n} \sum_{C_j\in\boldsymbol{C}} \omega_j \max\left\{1-\sum_{i\in I_j^+} x_i - \sum_{i\in I_j^-}(1-x_i), 0\right\} , \tag{3.4}$$

with the set of unweighted logical rules (namely to be hard-satisfied) integrated by linear constraints expressing their distance to satisfaction:

$$1-\sum_{i\in I_j^+} x_i - \sum_{i\in I_j^-}(1-x_i) \leq 0 .$$

We are now interested in defining a new kind of probabilistic graphical models, called *Hinge–Loss Markov Random Fields* (HL–MRFs) [7,8], considering a hinge–loss energy function. First to define these models, we note that any constraint distance to satisfaction corresponds to a hinge–loss function. Since the logical clauses can be converted into piecewise-linear constraints exploiting Łukasiewicz logic, in the following we will consider a more general form for the allowed constraints including arbitrary linear constraints. In particular, the problem in equation (3.4) can be generalized to

$$\operatorname*{arg\,min}_{y\in[0,1]^n} \sum_{C_j\in\boldsymbol{C}} \omega_j \max\left\{l_j(\boldsymbol{y}), 0\right\} ,$$

where l_j denotes any linear function of continuous random variables $\boldsymbol{y}$, each term in the sum denotes its distance from the satisfaction of $l_j(\boldsymbol{y}) \leq 0$ and the weight ω_j scales the distance from satisfaction of the corresponding constraint. We note that this definition applies to equality constraint as well, being representable as pair of inequalities.

Definition 3.3. *Let $\boldsymbol{y} = (y_1, \ldots, y_n), \boldsymbol{x} = (x_1, \ldots, x_{n'})$ be vector or variables with joint domain $\boldsymbol{D} = [0,1]^{n+n'}$ and $\boldsymbol{\phi} = (\phi_1, \ldots, \phi_m)$ a vector of continuous potentials such that*

$$\phi_j(\boldsymbol{y}, \boldsymbol{x}) = (\max\{l_j(\boldsymbol{y}, \boldsymbol{x}), 0\})^{p_j} \ , \tag{3.5}$$

where l_j is a linear function and $p_j \in \{1, 2\}$. Let $\boldsymbol{c} = (c_1, \ldots, c_r)$ be a vector of linear constraints (representing hard-constraints) defining the feasible set:

$$\tilde{\boldsymbol{D}} = \{(\boldsymbol{y}, \boldsymbol{x}) \in \boldsymbol{D} : \, c_k(\boldsymbol{y}, \boldsymbol{x}) \leq 0, \forall k \in \{1, \ldots, r\}\} \ .$$

Given a vector of nonnegative weights $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_m)$, for any $(\boldsymbol{y}, \boldsymbol{x}) \in \boldsymbol{D}$, a constrained hinge–loss energy function *$\boldsymbol{f}_\omega$ is defined as:*

$$\boldsymbol{f}_\omega(\boldsymbol{y}, \boldsymbol{x}) = \sum_{j=1}^{m} \omega_j \phi_j(\boldsymbol{y}, \boldsymbol{x}) \ .$$

We present the definition in a conditional form, however this gives a more general representation, indeed the set of conditioning variables may be empty.

Definition 3.4 (HL–MRFs)**.** *A* hinge-loss Markov random field *P over random variables $\boldsymbol{y}$ conditioned on random variables $\boldsymbol{x}$ is a probability density defined as follows:*

$$P(\boldsymbol{y}|\boldsymbol{x}) = \begin{cases} 0 & \textit{if } (\boldsymbol{y}, \boldsymbol{x}) \notin \tilde{\boldsymbol{D}} \\ \frac{1}{Z(\boldsymbol{\omega}, \boldsymbol{x})} \exp(-\boldsymbol{f}_\omega(\boldsymbol{y}, \boldsymbol{x})) & \textit{otherwise} \end{cases} ,$$

where

$$Z(\boldsymbol{\omega}, \boldsymbol{x}) = \int_{\boldsymbol{y}|(\boldsymbol{y}, \boldsymbol{x}) \in \tilde{\boldsymbol{D}}} \exp(-\boldsymbol{f}_\omega(\boldsymbol{y}, \boldsymbol{x})) d\boldsymbol{y} \ ,$$

and $\boldsymbol{f}_\omega$ is a hinge-loss energy function.

Whereas Markov Logic Networks define template for MRFs over boolean variables, Probabilistic Soft Logic (PSL) can be seen as a templating language for HL–MRFs, which are defined over continuous variables. In particular, a PSL program defines a class of HL–MRFs that are parameterized by the input data and where potential functions are associated to a set of templates corresponding to logical constraint. In PSL two types of rules can be considered to represent hinge–loss potential templates: *logical rules*, based on the mapping from logical clauses to hinge–loss potentials and *arithmetic rules* providing additional syntax to define constraints.

Definition 3.5. *A* PSL program *is a set of rules, each of which is a template for hinge–loss potentials or hard linear constraints. When a PSL program is grounded to a certain domain, it produces a HL–MRF conditioned on any specified observations.*

The rules in a PSL program are written in a first–order logical language. This means that the syntax of its logical rules involving *constants, variables* and *predicates* can be described as in Sec. 2.1.1, however PSL does not admit FOL functions except the ones defined in the arithmetical rules. We recall that the term *grounding* refers to the application of any predicate to its arguments consisting of only constants symbols. With the variables in the distribution grounded, each rule in the PSL program is applied to the inputs and produces hinge–loss potentials or hard linear constraints to be added to the HL–MRF. Any grounded logical rule has associated a potential as that one in equation (3.5) that can be taken as linear or quadratic. The presence of a weight determines if such rule has to be added to the HL–MRF with the weight as parameter or to the set of constraints defining the feasible set. For what concerns the logical expressiveness, PSL focuses on disjunctive clauses with possible non-negative weights associated, that are converted using the Łukasiewicz t-conorm $x \oplus y = \min\{x + y, 1\}$, even if some expedients can be considered to slightly increase its capability to deal with no-disjunctive formulas and negative rule weights [7].

In the following, we will define how PSL can be exploited to define both learning and inference machine learning tasks.

MAP Inference

One of the tasks we may carry out in PSL is *maximum a posteriori* (MAP) inference aiming at finding a most probable assignment to the free variables $\boldsymbol{y}$ given observations $\boldsymbol{x}$. Since in HL–MRFs, the partition function Z does not depend on $\boldsymbol{y}$, the exponential is maximized by minimizing its negated argument and the MAP problem can be defined as:

$$\begin{aligned} \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \equiv \arg\min_{\boldsymbol{y}} \boldsymbol{f}_{\omega}(\boldsymbol{y},\boldsymbol{x}) &= \arg\min_{\boldsymbol{y}} \boldsymbol{\omega}^T \cdot \boldsymbol{\phi}(\boldsymbol{y},\boldsymbol{x}) \\ \text{such that:} \qquad & c_k(\boldsymbol{y},\boldsymbol{x}) \leq 0,\ \forall k \in \{1,\dots,m\} \end{aligned} \tag{3.6}$$

MAP is a very fundamental task for PSL, indeed it allows the method to make predictions. In addition, as expressed by equation (3.6) this can be formulated as convex optimization and for instance some interior-point methods can be exploited. However, we also notice that for PSL has also been considered a new algorithm based on consensus optimization to scale to very large HL–MRFs for exact MAP inference. In addition, several steps of MAP inference can be required to perform the weight learning of the constraints that are involved in the inference process, according to an iterative procedure.

Rule Weights Learning

A possible way to carry out weight learning in a HL–MRFs, is by maximizing the likelihood of the training data. In particular, we can maximize the log–likelihood of the training data via gradient descent, where the derivative with respect to any rule weight ω_j is expressed by

$$\frac{\partial \log P(\boldsymbol{y}|\boldsymbol{x})}{\partial \omega_j} = E_{\boldsymbol{\omega}}\left[\phi_j(\boldsymbol{y},\boldsymbol{x})\right] - \phi_j(\boldsymbol{y},\boldsymbol{x}) \ ,$$

where $\mathbb{E}_{\boldsymbol{\omega}}$ denotes the expectation under the distribution defined by $\boldsymbol{\omega}$. The gradient with respect to the j–th clause weight is null when the distance from satisfaction of the j-th constraint on the training data $\boldsymbol{y}_t$ corresponds to what is predicted by the model: $\phi_j(\boldsymbol{y}_t) = E_{\boldsymbol{\omega}}\left[\phi_j(\boldsymbol{y},\boldsymbol{x})\right]$. Computing the expected value is intractable in a general setting and improving PSL weight learning is an open research problem. A common solution is to approximate the expected value with the most probable approximation (MAP solution) according to the current weights: $E_{\boldsymbol{\omega}}\left[\phi_j(\boldsymbol{y}, \boldsymbol{x})\right] \approx \phi_j(\boldsymbol{y}_M)$, where $\boldsymbol{y}_M$ denotes the MAP solution at the current weights. Under this assumption, weight learning becomes tractable, because inference to determine the most probable interpretation corresponds to solving the convex optimization task as previously described.

3.4 Semantic–Based Regularization

Markov Logic Networks and Probabilistic Soft Logic provide a generic AI interface layer for machine learning by implementing a probabilistic logic. However, the integration with the underlying learning processes working on the low-level sensorial data is shallow: a low-level learner is trained independently, then frozen and stacked with the AI layer providing a higher-level inference mechanism. In this section we present *Semantic–Based Regularization* (SBR) [34], a language proposed to directly improve the underlying learner, while also providing the higher-level integration with logic. A strong connection between SBR and MLNs has also been pointed out in [34]. In particular, SBR can be seen as a MLN where the FOL formulas and node values are replaced by their fuzzy generalization with the node values computed by kernel machines. More generally, SBR is a unified framework for inference and learning centered around the notion of constraint. On the one hand, the framework can exploit different machine learning techniques to learn from continuous feature-based representations some relations among patterns, e.g. in case of *Kernel Machines*. On the other hand, SBR converts a set of first–order logic (FOL) formulas expressing some prior knowledge on the task in a set of functional constraints according to a fuzzy logic translation of formulas.

In particular, SBR builds a multi-layer architecture where at the first layer kernel machines extract high-level feature representation of the input data. Then a second layer takes as input the output of the kernel machines and implements a fuzzy conversion of the FOL formulas in the knowledge base. The resulting model is continuous and the semantic inference provided by the logic rules can be back-propagated down to the kernel machines using any gradient-based schema. In particular, the logical layer allows us to exploit possible available unsupervised data improving the capacity of generalization of the model and can be exploited to correct possible mistakes from the kernel machines layer. In the following we sketch how the conversion, from FOL to fuzzy logic and then again to functional constraints, is carried out in SBR, however we will provide more details on this general process in Sec. **??**.

Let us consider a set of predicate functions $\mathbf{P} = \{p_1, \ldots, p_J\}$, where each p_j with arity $a_j > 0$ is grounded from the set $\mathcal{X}_j = \mathcal{X}_{j1}, \ldots, \mathcal{X}_{ja_j}$. In the following, we indicate by $p_j(\mathcal{X}_j)$ the set of possible groundings for the j–th predicate and $\mathcal{P}(\mathcal{X}) = p_1(\mathcal{X}_1) \cup \ldots \cup p_J(\mathcal{X}_J)$. Assuming all the predicates are evaluated in $[0, 1]$, then the truth degree of a formula containing an expression E can be computed by fuzzy logic operators according to Table 2.2. The universal and existential quantifiers over a variable x_i are converted according to the following expressions:

$$\begin{aligned} \forall x_i E\big(\mathcal{P}(\mathcal{X})\big) \quad &\Longrightarrow \quad \Phi_\forall\big(\mathcal{P}(\mathcal{X})\big) = \min_{\boldsymbol{x}_i \in \mathcal{X}_i} t_E\big(\mathcal{P}(\mathcal{X})\big) \ , \\ \exists x_i E\big(\mathcal{P}(\mathcal{X})\big) \quad &\Longrightarrow \quad \Phi_\exists\big(\mathcal{P}(\mathcal{X})\big) = \max_{\boldsymbol{x}_i \in \mathcal{X}_i} t_E\big(\mathcal{P}(\mathcal{X})\big) \ . \end{aligned}$$

However, for implementation reasons, the universal quantifier is often translated as an arithmetic mean over a variable domain, namely as:

$$\forall x_i E\big(\mathcal{P}(\mathcal{X})\big) \quad \Longrightarrow \quad \Phi_\forall\big(\mathcal{P}(\mathcal{X})\big) = \frac{1}{|\mathcal{X}_i|} \sum_{\boldsymbol{x}_i \in \mathcal{X}_i} t_E\big(\mathcal{P}(\mathcal{X})\big) \ .$$

To summarize, the network encoded by SBR for a given KB can be defined as in the following.

Definition 3.6 (SBR network). *Let us consider a set of FOL formulas in a knowledge base KB composed by predicates that are grounded by a set of constants. In particular, we denote by $\boldsymbol{x}$ the feature vector associated to the input variable x, and by f_i the function implemented by a Kernel Machine to approximate the i–th unknown predicate p_i. SBR builds a multi-layer network computing the fuzzy FOL approximation of KB, where the value $f_i(\boldsymbol{x})$ replaces a grounded unknown predicate $p_i(x)$.*

We consider a multi-task learning problem, where a set of J unknown functions have to be estimated and another J' functions are known a priori, where $\boldsymbol{f} = (f_1, \ldots, f_J, \ldots, f_{J+J'})$ denotes the vector of such functions. We assume we are given a set of FOL formulas $\varphi_1, \ldots, \varphi_H$ with corresponding fuzzy conversion $\Phi_1, \ldots, \Phi_H$. Then the formulas can be enforced to be satisfied by requiring $1 - \Phi_h(\boldsymbol{f}) = 0$, $0 \leq \Phi_h(\boldsymbol{f}) \leq 1$, with $h = 1, \ldots, H$. The functionals Φ_h can express a property of a single function or correlate multiple functions in order to support the learning process. Assuming the function f_j in $\boldsymbol{f}$ belongs to a certain functional space $\mathcal{H}_j$[2], we can also express a regularization term according to the parsimony principle. Following the classical penalty approach, the learning problem can be defined as constrained optimization by requiring the minimization of the following cost function.

$$C[\boldsymbol{f}] = \sum_{j=1}^{J} ||f_j||^2_{\mathcal{H}_j} + \sum_{h=1}^{H} \lambda_h (1 - \Phi_h(\boldsymbol{f})) ,$$

where λ_h is the weight of the h–th constraint expressing which constraints are more costly if violated. The constraints are generally enforced only over a finite sample of input values. If we denote by $\boldsymbol{f}(\mathcal{X})$ the set of all the possible grounding of functions on the overall sample $\mathcal{X}$, we get the following cost function:

$$C[\boldsymbol{f}(\mathcal{X})] = \sum_{j=1}^{J} ||f_j||^2_{\mathcal{H}_j} + \sum_{h=1}^{H} \lambda_h (1 - \Phi_h(\boldsymbol{f}(\mathcal{X}))) . \tag{3.7}$$

However, the cost function in equation (3.7) is in general not convex, when the functional constraints Φ_H can represent arbitrary FOL formulas according to a t-norm fuzzy logic semantics to be chosen (validated), e.g. Gödel, Łukasiewicz or Product logic. The learning framework we will present in the next chapters can be considered as an extension of SBR. However in Sec. 4.1 we provide a fragment of Łukasiewicz logic yielding convex functional constraints. This means that the cost function in equation (3.7) becomes tractable once the FOL formulas are converted with the **Ł**–convex fragment.

3.5 Logic Tensor Networks

A Logic Tensor Network (LTN) [137] provides an integration between first–order logic and learning based on neural networks, all implemented in TensorFlow. In particular, LTN exploits *neural tensor networks* (NTN) [139] to

[2]In particular, here $\mathcal{H}_j$ is assumed to be a Reproducing Kernel Hilbert Space (RKHS), therefore any $f \in \mathcal{H}$ can be represented by a certain kernel function.

evaluate all the predicates involved in the formulas on the training data (a collection of real-valued features), while SBR generally exploits kernel machines. Once all the grounded predicates are computed, these values are combined according to a certain fuzzy logic and then aggregated in an overall loss function defining the satisfiability error to be minimized.

Since LTN and SBR exploit t-norm fuzzy logics in the same way to define an optimization problem, we only add some comments about the chosen architecture of LTN to compute the grounded predicates, i.e. the neural tensor network. More formally, let us consider a set of patterns $v_1, \dots, v_m \in \mathbb{R}^n$ collected in a vector $\boldsymbol{v} \in \mathbb{R}^{nm}$ and an m–ary predicate P. The grounding of P, corresponding to the truth evaluation of P, can be defined as

$$\mathcal{G}(P) = \sigma\left(u_P^T \tanh\left(\boldsymbol{v}^T W_P^{[1:k]} \boldsymbol{v} + V_P \boldsymbol{v} + B_P\right)\right) \;, \tag{3.8}$$

where $W_P^{[1:k]}$ is a 3-D tensor in $\mathbb{R}^{nm \times nm \times k}$, V_P is a matrix in $\mathbb{R}^{k \times nm}$, $B_P \in \mathbb{R}^k$, and σ denotes the sigmoid function. In particular, the grounding of P in equation (3.8) is defined as a generalization of the neural tensor network, that has already been shown to provide good results by exploiting simple logical constraints in knowledge compilation [139]. It is worth noticing that, since a NTN replaces a standard linear neural network layer with a bilinear tensor layer, the input feature vectors can be combined across multiple dimensions. Indeed, each tensor slice can be seen as mediating the relationship on the input differently.

Chapter 4

Convex Logical Constraints

The theoretical results we present in this chapter may be exploited in different learning settings, especially in contexts where some relational knowledge on the task to be learned is available. As we already noticed in Sec. 2.4.4, a learning process can be thought of as a constraint satisfaction problem, where the constraints represent the knowledge about the functions to be learned. In particular, in multi-task learning beside the supervised examples additional information on the targets can be expressed as abstract knowledge by logical constraints. Supervisions act as a special class of constraints providing positive as well as negative examples for the task, while it may be also useful to express relationships among the unknown functions. For example, in a classification problem, we may be interested in the satisfaction of a rule like *"any pattern classified as a cat has to be classified as an animal"*, where *cat* and *animal* have to be thought of as the membership functions of two classes to learn. In such a sense, symbolic logic provides a natural way to express factual and abstract knowledge about a problem by means of logical formulas in a certain logic. Logical representations have been successfully employed in various learning schemes from a long time, since they allow high-level representations. In addition, we can exploit theorems and fundamental properties of the chosen logic to get an advantage for the learning strategy.

Framework. Let us consider a set of predicates $\mathbf{P} = \{p_1, \ldots, p_J\}$, all collected in the vector $\boldsymbol{p} = (p_1, \ldots, p_J)$, with possibly different domains $\mathcal{D}_j \subseteq \mathbb{R}^{n_j}$, where $n_j \in \mathbb{N}$. Any predicate p_j has associated a certain arity $a_j \in \mathbb{N}$ and its domain can be decomposed as $\mathcal{D}_j = \mathcal{D}_{j1} \times \ldots \times \mathcal{D}_{ja_j}$ for opportune $\mathcal{D}_{ji}$. Given a certain $\bar{x} \in \mathcal{D}_j$, the evaluation of p_j on $\bar{x}$, i.e. $p_j(\bar{x}) \in [0, 1]$, is said a *grounding* of p_j, while $p_j(\mathcal{D}_j)$ denotes the vector of all the groundings of p_j on its domain. For simplicity, we also write $\boldsymbol{p}(\mathcal{D}) = (p_1(\mathcal{D}_1), \ldots, p_J(\mathcal{D}_J))$

to indicate the vector of all the grounded predicates.

In a given learning problem, any predicate is implemented by a model depending on parameters, like a kernel machine, a neural network, and so on, restricted to map into a $[0,1]$–value. Even if we do not mention function symbols of FOL so far, the representation we are giving remains exactly the same provided a set of functions $\mathbf{F} = \{f_1, \dots, f_K\}$ defined among the domains of discourse and collected in the vector denoted by $\boldsymbol{f}$. If function symbols are taken into account, the models implementing them have to be indicated as well. In an experimental setting, the predicates (and functions) never will be evaluated on their whole domains, but just on a finite subset, the available examples. In the following we denote by sets $\mathcal{X}_i$ the available samples for the arguments of the involved predicates and by $\boldsymbol{p}(\mathcal{X})$ the overall vector of the groundings on the whole dataset. For what concerns the connectives and quantifiers, they are threated using the fuzzy generalization of first–order logic that was first proposed by Novak [114]. In particular, a *t-norm fuzzy logic* (as defined in Sec. 2.2) generalizes Boolean logic to variables assuming values in $[0,1]$ and is defined by its t-norm modeling the logical AND. Some possible implementations of the connectives are reported in Table 2.2, while we notice that the universal and existential quantifiers can be considered as a fuzzy conjunction and disjunction, respectively. For the moment, we suppose to convert the quantifiers as the minimum and maximum operations that are common to any t-norm fuzzy logic, however in Sec. 4.2 we will discuss in detail the general case of fuzzy aggregation functions (see also Sec. 2.3). For instance, consider a quantifier-free FOL formula $\varphi(x_i)$, with fuzzy conversion Φ, depending on a certain variable $x_i \in \mathcal{X}_i$. In this case, the result of quantifying x_i in φ is converted according to the following rules.

$$\begin{aligned} \forall x_i\, \varphi(x_i) &\Longrightarrow \min_{x_i \in \mathcal{X}_i} \Phi(x_i, \boldsymbol{f}, \boldsymbol{p}) \\ \exists x_i\, \varphi(x_i) &\Longrightarrow \max_{x_i \in \mathcal{X}_i} \Phi(x_i, \boldsymbol{f}, \boldsymbol{p}) \end{aligned} \tag{4.1}$$

When multiple quantified variables are present, the conversion is recursively performed from the outer to the inner variables.

In a given learning problem, where all the knowledge consists in a set of FOL formulas $KB = \{\varphi_1, \dots, \varphi_H\}$, we suppose that some of the elements (individuals, functions or predicates) are unknown. The learning process aims at finding a good approximation of each unknown element, so that the estimated values will satisfy the formulas for the input samples. Since the formulas in KB are evaluated into $[0, 1]$, and a formula is true if it is evaluated as 1, in order to satisfy the constraints we may minimize the following loss function

$$L(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p}) = \sum_{h=1}^{H} \lambda_h \big(1 - \Phi_h(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})\big) \tag{4.2}$$

where any λ_h is a weight for the h–th logical constraint and any formula is indicated as depending on the whole dataset $\mathcal{X}$ for clarity. The weights of the constraints may be either evaluated with respect to the performance on the learning problem we want to solve or learned as well. Here, we only remark that several strategies for learning the weights associated to the rules have been considered in the literature [83, 85, 153] and that in general, it is not a trivial task how to determine them. In general it has been considered both methods to learn the structure of the rules and to learn their weight (or confidence). A deeper analysis of this wide topic is beyond the scope of this thesis, however we recommend [30] for a survey.

About Convexity Previously, we described how to convert FOL formulas into functional constraints yielding an overall cost functional. Minimizing this loss allows us to learn the optimal values for the parameters of the models and this process can be tackled via several optimization techniques. One desired property of a cost functional is to be convex. Indeed, non-convex optimization is intractable in a general sense [141] and it can be efficiently faced only when sub-optimal solutions are acceptable[1]. On the other hand, convex optimization is very well understood and several algorithms are available to efficiently find optimal solutions. For example, Conjugate Gradient Descent [104] is guaranteed to converge to an optimal solution in linear time when the cost functional is convex. Many methods in machine learning explicitly aim at defining convex cost functionals by constraining the form of the approximation. For example, training of support vector machines [21] corresponds to solving a quadratic (with linear constraints) optimization problem.

[1]An optimal solution is a solution for which there are no other solutions providing a lower cost. In convex optimization, any sub-optimal solution turns out to be an optimal one, indeed any local minimum for a convex function is global as well.

However, it is not obvious how to get a convex cost functional in general. For instance, the definition of a convex cost functional is more challenging when the learning task is more complex and general logic knowledge needs to be exploited to express properties of the unknown function, like assumed by statistical relational learning (SRL) methods. SRL learners includes methods like Markov Logic Networks, however, inference in MLNs defines a non-convex optimization problem, making the methodology impractical for large learning problems. To overtake this limitation, probabilistic soft logic (PSL) relaxes the learning task using fuzzy logic and restricts the knowledge base to clauses with a specific form. Under these constraints inference corresponds to a convex optimization problem, which can be solved even for large learning tasks. However, the limitations in the form of the clauses prevents the application of PSL to arbitrary learning tasks. In the next section, we study under which conditions a general description of the knowledge base corresponds to a convex optimization problem, when integrated into learning. This class of descriptions is larger than that considered in PSL and similar learning methods, making this theoretical result useful across a wide range of machine learning applications.

4.1 The Convex Łukasiewicz Fragment

Given a set of first–order logic constraints, equation 4.2 provides a methodology to define a cost functional that can be minimized to determine the unknown objects involved in a certain learning problem. As we already pointed out above, we are particularly interested in obtaining a loss function that is convex and this strongly depends on the specific translation of the connectives occurring in the formulas as well as on the chosen implementations of the learnable functions. This is the reason why we investigate different ways of translating the logical connectives. However, in this section we provide a logical fragment of propositional Łukasiewicz logic whose corresponding functional constraints turn out to be convex and, as we will see in the next sections, the fragment turns out to be not extendible if we want to preserve convexity. Since the result is about logic, it applies to a wide class of learning settings exploiting logical arguments. In particular, we discuss the general case of a multi-task learning problem, where we are interested in the integration of prior knowledge with supervisions for learning the set $\mathbf{P} = \{p_1, \ldots, p_J\}$ of predicates. For simplicity, in the remaining of this chapter we suppose we are not given FOL functions, namely $\mathbf{F} = \emptyset$.

Although in principle the logical constraints may be expressed by any (fuzzy) logic, there are several reasons to work into the Łukasiewicz frame like other authors do [6, 136], especially to investigate convexity. For instance, among the three fundamental fuzzy logics given by a continuous t-norm (Łukasiewicz, Gödel and Product), the Łukasiewicz logic is the only one providing an equivalent *prenex normal form* (i.e. quantifiers followed by a quantifier-free part) and a continuous involutive negation ($\neg\neg x = x$) preserving the De Morgan laws. It is also worth to notice that functions corresponding to Product t-norm are at most *quasi concave*, while the Gödel t-norm is included in Łukasiewicz logic, indeed it is represented by the Łukasiewicz weak conjunction. Finally, the McNaughton Theorem provides a functional representation of Łukasiewicz formulas by piecewise linear functions and then, we can restrict the study of convexity to such kind of functions. Exploiting this result, we are able to characterize the fragment of Łukasiewicz formulas corresponding to convex functional constraints. Formulas belonging to this fragment are referred as *simple Łukasiewicz clausal forms* in the literature and, among others, the satisfiability problem for these formulas has shown to have linear-time complexity [11].

4.1.1 Propositional Łukasiewicz Logic

In Sec. 2.2 we briefly introduced the fuzzy logic realm and the fundamental notions of t-norms and other fuzzy connectives. However, the results of this chapter rely in particular on Łukasiewicz logic **Ł** whose main properties are described in this section. **Ł** is the fuzzy logic we get if we assume the t-norm $x \otimes y = \max\{0, x + y - 1\}$ as truth function for the conjunction on the continuous values $[0, 1]$. It is worth noticing that, although for the learning settings we exploit a first–order logic language, the objective functions are evaluated on finite training sets and according to the following rules, FOL formulas can be rewritten in an equivalent quantifier-free form. Indeed, for any predicate p occurring in formulas and evaluated on a sample $\mathcal{X}_p$ we get:

$$\forall x\, p_i(x) \simeq \bigwedge_{a\in\mathcal{X}_p} p_i(a), \quad \exists x\, p_i(x) \simeq \bigvee_{a\in\mathcal{X}_p} p_i(a). \tag{4.3}$$

This process can be carried out for every predicate in any FOL formula, until we are given an equivalent propositional formula. As a result, we can exploit stronger results from propositional theories still yielding FOL expressiveness.

Propositional Łukasiewicz logic is sound and complete with respect to its standard algebra on $[0,1]$ and the operations corresponding to Łukasiewicz connectives are reported in Table 4.1. The connectives $\otimes, \oplus$ correspond to

Formula	**Operation**
conjunctions	
$x \otimes y$	$\max\{0, x+y-1\}$
$x \wedge y$	$\min\{x, y\}$
disjunctions	
$x \oplus y$	$\min\{1, x+y\}$
$x \vee y$	$\max\{x, y\}$
implications and negation	
$x \rightarrow y$	$\min\{1, 1-x+y\}$
$x \leftrightarrow y$	$1-\lvert x-y\rvert$
$\neg x$	$1-x$
constants	
$\bar{0}$	0
$\bar{1}$	1

Table 4.1: *Lukasiewicz connectives and their algebraic counterparts.*

Łukasiewicz t-norm and t-conorm. They are called *strong* connectives in opposition to $\wedge, \vee$ that are called *weak*. Indeed, for all $x, y \in [0,1]$, the following relations hold according to the order defined in equation (2.1):

$$x \otimes y \le x \wedge y \le x \vee y \le x \oplus y.$$

The implication is the residuum of the t-norm and in **Ł** it can be defined as $x \rightarrow y := \neg x \oplus y$, while the double implication as $x \leftrightarrow y := (x \rightarrow y) \otimes (y \rightarrow x)$.

In addition, $\bar{0} \equiv x \otimes \neg x$ and $\bar{1} \equiv x \oplus \neg x$ for any $x \in \mathcal{V}$, where $\mathcal{V}$ denotes the set of all the propositional variables. It is worth to notice that in **Ł** we have two conjunctions and two disjunctions (a strong and a weak). Weak connectives can be defined from the strong ones in every fuzzy logic (see equation 2.2). Although the two conjunctions and the two disjunctions coincide on the crisp values $\{0, 1\}$, they generalize quite differently on the continuous values $[0, 1]$.

Boolean logic has several fundamental properties that allow us to easily manipulate the formulas, however in the fuzzy logic realm things are slightly more complicated. For instance, the *De Morgan laws* between both weak and strong connectives hold, as well as the *Distributive laws* of strong over weak. We only notice that the distributive property does not hold between strong conjunction and strong disjunction, namely for some $x, y, z \in [0, 1]$

$$x \oplus (y \otimes z) \not\equiv (x \oplus y) \otimes (x \oplus z),$$

as shown in the following counter-example taking $x = 0.1$, $y = z = 0.5$:

$$\min\{1, x + \max\{0, y + z - 1\}\} = 0.1$$

$$\max\{0, \min\{1, x + y\} + \min\{1, x + z\} - 1\} = 0.2.$$

In view of the learning procedure, formulated as an optimization problem, we are interested in a *functional representation* of logical formulas. Indeed, FOL will be translated into functional constraints for the objective functions, as sketched in equation (4.2). Since the algebra of **Ł**–formulas on n variables is isomorphic to the algebra of functions from $[0, 1]^n$ to $[0, 1]$ as shown in [114], we can translate formulas into functions according to equation (2.3). The fundamental result about the functional representation of **Ł**–formulas is given by the already mentioned McNaughton Theorem. It states that, for the propositional case, the functions corresponding to Łukasiewicz formulas are McNaughton functions defined on $[0, 1]^n$, namely continuous piecewise linear functions with integer coefficients. As a consequence, for every **Ł**–formula φ depending on n propositional variables, we can consider its corresponding function $f_\varphi : [0, 1]^n \to [0, 1]$, whose value on each point is exactly the evaluation of the formula with respect to the same variable assignment. Hence we can investigate the convexity of such functions that, for the McNaughton Theorem, are McNaughton functions.

4.1.2 Convex McNaughton Functions

In Łukasiewicz logic are definable different operations and we are interested in the as large as possible set of **Ł**–formulas corresponding to convex McNaughton functions. We start with the investigation of logical connectives corresponding to operations that preserve concavity (convexity) aiming at the identification of a whole concave (convex) fragment. However for clarity, we first recall the definition of a concave (convex) function.

Definition 4.1 (Concave and Convex Functions)**.** *Let us consider a function* $f : X \subseteq \mathbb{R}^n \to \mathbb{R}$*, f is said to be*

$$\begin{array}{rll} \textit{convex} & \textit{iff} & f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y) \\ \textit{concave} & \textit{iff} & f(\lambda x + (1-\lambda)y) \geq \lambda f(x) + (1-\lambda)f(y) \end{array}$$

for any $x, y \in X$*,* $\lambda \in [0,1]$.

The concavity and convexity of Łukasiewicz connectives have also been studied in [96], in which are identified the **Ł**–formulas that may be used to get concave payoff functions. In our framework, formulas built up from the concave fragment derive convex functional constraints and we also prove that they coincide with the whole class of concave Łukasiewicz functions.

Lemma 4.1. *Let* φ, ψ *be two* **Ł**–*formulas, then*

1. f_φ *is convex if and only if* $f_{\neg\varphi}$ *is concave;*
2. *if* f_φ, f_ψ *are concave then the functions* $f_{\varphi\wedge\psi}$ *and* $f_{\varphi\oplus\psi}$ *are concave;*
3. *if* f_φ, f_ψ *are convex then the functions* $f_{\varphi\vee\psi}$ *and* $f_{\varphi\otimes\psi}$ *are convex.*

Proof. We provide a point-by-point proof of this Lemma.

1. This is obvious since the opposite of any convex function is a concave one and vice versa.
2. If f_φ and f_ψ are concave, then for all x, y, $\lambda \in [0,1]$, $f_{\varphi\wedge\psi}(\lambda x + (1-\lambda)y) = \min\{f_\varphi(\lambda x + (1-\lambda)y),\ f_\psi(\lambda x + (1-\lambda)y)\} \geq \min\{\lambda f_\varphi(x) + (1-\lambda)f_\varphi(y),\ \lambda f_\psi(x) + (1-\lambda)f_\psi(y)\} \geq \lambda f_{\varphi\wedge\psi}(x) + (1-\lambda)f_{\varphi\wedge\psi}(y)$.
   Moreover, by definition $f_{\varphi\oplus\psi}(x) = \min\{1, f_\varphi(x)+f_\psi(x)\}$, thus if $f_{\varphi\oplus\psi}(\lambda x+$

$(1-\lambda)y) = 1$ then obviously it is greater or equal than $\lambda f_{\varphi\oplus\psi}(x) + (1-\lambda)f_{\varphi\oplus\psi}(y)$. Otherwise $f_{\varphi\oplus\psi} = f_\varphi + f_\psi$ and sum preserves concavity (and it preserves convexity too) so the thesis easily follows.

3. This point follows from 1) and 2) plus recalling that $f_{\varphi\vee\psi} = f_{\neg(\neg\varphi\wedge\neg\psi)}$ and $f_{\varphi\otimes\psi} = f_{\neg(\neg\varphi\oplus\neg\psi)}$.

□

As a consequence, the operations corresponding to the connectives $\wedge$, $\oplus$ preserve concavity, while the ones corresponding to $\vee$, $\otimes$ preserve convexity. In the following definition, we fix two different fragments of Łukasiewicz formulas which are built according to such connectives.

Definition 4.2. *Let $(\wedge,\oplus)^*$ and $(\otimes,\vee)^*$ be the smallest sets of formulas (up to equivalence) such that:*

- $\bar{0} \in (\wedge,\oplus)^*$ *and* $\bar{1} \in (\otimes,\vee)^*$*;*
- *if $x \in \mathcal{V}$, then $x, \neg x \in (\wedge,\oplus)^*$ and $x, \neg x \in (\otimes,\vee)^*$;*
- *if $\varphi_1, \varphi_2 \in (\wedge,\oplus)^*$, then $\varphi_1 \wedge \varphi_2$, $\varphi_1 \oplus \varphi_2 \in (\wedge,\oplus)^*$;*
- *if $\varphi_1, \varphi_2 \in (\otimes,\vee)^*$, then $\varphi_1 \otimes \varphi_2$, $\varphi_1 \vee \varphi_2 \in (\otimes,\vee)^*$.*

Anticipating the main result of this section, we refer to $(\wedge,\oplus)^*$ as *the concave fragment* and to $(\otimes,\vee)^*$ as *the convex fragment* of Łukasiewicz logic. Since $\bar{0}, \bar{1}$ correspond to constant functions and the literals correspond to projections or their negations, which are affine functions and hence both concave and convex, the formulas inside a specific fragment are guaranteed to be concave or convex respectively.

In order to prove that formulas in the concave (convex) fragment are the only ones corresponding to concave (convex) McNaughton functions, the following theorem plays a crucial role.

Theorem 4.1. *Any convex (concave) piecewise linear function on $\mathbb{R}^n$ can be expressed as a max (min) of a finite number of affine functions.*

Proof. See e.g. Theorem 2.49 pag.68 of [123] for a proof of this result. □

This means that, for each n–ary convex McNaughton function f there exist $M_1, \dots, M_k \in \mathbb{Z}^n$, $q_1, \dots, q_k \in \mathbb{Z}$ such that:

$$\text{for all } x \in [0,1]^n \qquad f(x) = \max_{i=1,\cdots,k} \left(M_i' \cdot x + q_i\right). \tag{4.4}$$

On the other hand, every concave McNaughton function can be expressed as the minimum of a finite number of affine functions. We only mention that the coefficients of the affine functions are constructively determined by the shape of the considered formula.

Example 4.1. *Let us consider $\varphi = ((x \wedge y) \oplus \neg y \oplus z) \wedge \neg z$, then $\varphi \in (\wedge, \oplus)^*$ and it is equivalent to $(x \oplus \neg y \oplus z) \wedge (y \oplus \neg y \oplus z) \wedge \neg z$. From this latter expression, we get:*

$$f_\varphi(x, y, z) = \min\{1, x - y + z + 1, 1 - z\}.$$

Finally, exploiting equation (4.4) and thanks to Lemma 4.1, we can prove that $(\wedge, \oplus)^*$ and $(\otimes, \vee)^*$ coincide with the sets of all **Ł**–formulas whose corresponding McNaughton functions are concave and convex respectively.

Proposition 4.1. *Let $f_\varphi : [0,1]^n \to [0,1]$ be a McNaughton function. Then,*

1. *f_φ is concave if and only if $\varphi \in (\wedge, \oplus)^*$;*
2. *f_φ is convex if and only if $\varphi \in (\otimes, \vee)^*$.*

Proof. First of all we note that, as a consequence of Lemma 4.1, if φ belongs to the concave fragment, then f_φ is a concave function. Indeed, all the connectives occurring in φ correspond to operations that preserve concavity and literals and constants correspond to affine functions. The same argument holds if the formula belongs to the convex fragment.

On the other hand, let us suppose that f_φ is a concave piecewise linear function, hence there exist some elements $a_{i_j}, b_i \in \mathbb{Z}$ for $i = 1, \dots, m$ and $j = 1, \dots, n$, such that:

$$f_\varphi(x) = \min_{i=1}^{m} a_{i_1}x_1 + \ldots + a_{i_n}x_n + b_i, \qquad x \in [0,1]^n.$$

If we set $p_i(x) = a_{i_1}x_1 + \ldots + a_{i_n}x_n + b_i$ for $i = 1, \dots, m$, our claim follows provided every p_i corresponds to a formula in $(\wedge, \oplus)^*$. Indeed the operation of minimum is exactly performed by the connective $\wedge$. Let us fix $i \in \{1, \dots, m\}$, then we can write

$$p_i(x) = \sum_{j \in P_i} a_{i_j} x_j + \sum_{j \in N_i} a_{i_j} x_j + b_i,$$

where $P_i = \{j \leq n : a_{i_j} > 0\}$ and $N_i = \{j \leq n : a_{i_j} < 0\}$. For short, given any $\psi \in$ Ł, we write $a\psi$ with the meaning of $\bigoplus_{i=1}^{a} \psi$ or $\bar{0}$, if $a > 0$ or $a = 0$ respectively. Therefore, we can consider the following formula as corresponding to the function p_i:

$$\varphi_i = \bigoplus_{j \in P_i} a_{i_j} x_j \oplus \bigoplus_{j \in N_i} |a_{i_j}| \neg x_j \oplus q_i \bar{1}.$$

Indeed the first strong disjunction corresponds to all the positive monomials of p_i. The second one corresponds to all the negative monomials of p_i, but it also introduces the quantity $\sum_{j \in N_i} |a_{i_j}|$. Finally $q_i = b_i - \sum_{j \in N_i} |a_{i_j}|$, with $q_i \geq 0$ since $p_i(x) \geq 0$ for all $x \in [0,1]^n$ and in particular $p_i(\bar{x}) = b_i - \sum_{j \in N_i} |a_{i_j}| \geq 0$ where $\bar{x}$ is the vector with 0 in positive and 1 in negative monomial positions respectively. The overall formula can be written as $\varphi = \varphi_1 \wedge \ldots \wedge \varphi_m$. $\square$

Summing up, the largest fragments of Łukasiewicz logic whose McNaughton functions are either concave or convex are determined. This result is very general because it can be applied to different learning settings, where the use of the fragment brings benefits to the solution. In particular, in Chap. 5, we show how to exploit this result into kernel machines, collective classification and Probabilistic Soft Logic theories. Finally, it is worth to notice that in the literature, logical constraints are often initially expressed in boolean form and then *fuzzified*. Since the fragments we define contain both a conjunction and a disjunction which are coherent with boolean connectives on the crisp values, in principle one can almost always translate boolean formulas into convex Łukasiewicz constraints. However, as we sketch in the next subsection, any choice for this translation can slightly modify the expressiveness of the formulas we are dealing with.

4.1.3 Notes on Expressiveness

As already noticed, in practical problems we are often given a set of boolean formulas. Hence we have to decide a suitable way of translating them into the language of fuzzy logics, and in particular Łukasiewicz logic, preserving their initial intention. The majority of authors [13, 34, 137] convert boolean conjunction with the t-norm and the disjunction with the t-conorm, namely with the pair $(\otimes, \oplus)$. It is worth noticing that Proposition 4.1 characterizes the concave and the convex formulas in **Ł**, but in general it does not provide an effective way to embed any boolean formula into a specific fragment. However, this is always guaranteed by making use of either the concave or the convex connectives if we consider any boolean formula without implication and with negation on at most literals. In particular, given any boolean formula, we can always rewrite it into a *conjunctive normal form* (CNF) and then we can apply the following two translations to the conjunctions and disjunctions to fall into the concave or the convex fragment respectively:

$$\text{(concave)} \quad \bigwedge_{i=1}^{n} \bigoplus_{j=1}^{m} (\neg) l_{ij}, \qquad \text{(convex)} \quad \bigotimes_{i=1}^{n} \bigvee_{j=1}^{m} (\neg) l_{ij}. \tag{4.5}$$

Actually, there are several possibilities to translate the boolean connectives into the Łukasiewicz ones, where we have two conjunctions and two disjunctions. Even if the weak and the strong operations coincide on the crisp values $\{0, 1\}$, the way they generalize on continuous values $[0, 1]$ determines different semantics for the resulting formulas. In the following we show some examples to compare possible representations.

Example 4.2 (Distributivity)**.** *In general, we can lose consistency on what we have to represent if we manipulate the boolean expressions before translating them into fuzzy terms. For instance, in boolean logic, the formulas $x \wedge (y \vee z)$ and $(x \wedge y) \vee (x \wedge z)$ are equivalent. However, if we translate them with the fragment $(\otimes, \vee)^*$ the equivalence still holds, whereas in general the same is not true with the pair $(\otimes, \oplus)$.*

A well-studied class of formulas is given by the Horn clauses, i.e. formulas with a propositional variable implied by a conjunction of propositional variables. They are very common in the literature since they are quite expressive and easy to be managed.

Example 4.3 (Horn Clauses)**.** *Given a Horn clause, if we translate the conjunction with the t-norm, then we get:*

$$(x_1 \otimes \ldots \otimes x_m) \rightarrow y \equiv (\neg x_1 \oplus \ldots \oplus \neg x_m \oplus y) \in (\wedge, \oplus)^*,$$

namely the class of Horn clauses translated into **Ł** *with* $\otimes$ *is strictly contained in the concave fragment. Indeed, the following formulas are in* $(\wedge, \oplus)^*$ *and do not correspond to Horn clauses:*

$$x \wedge y, \qquad x \oplus (y \wedge z), \qquad x \wedge (x \rightarrow y), \qquad (x \otimes y) \rightarrow (x \wedge z).$$

Finally in the rest of this section, we discuss some examples representing well-known rules in learning from logical constraints.

Example 4.4 (Manifold Regularization)**.** *Manifold regularization assumes that a given binary predicate* $R(a, b)$ *states when two objects* a *and* b *belong to the same manifold, requiring that the value of a predicate* P *should be consistent when evaluated on the two elements. This condition can be expressed by the following boolean formula:*

$$R(a, b) \rightarrow (P(a) \leftrightarrow P(b)), \tag{4.6}$$

where $P(a) \leftrightarrow P(b) \equiv (P(a) \rightarrow P(b)) \wedge (P(b) \rightarrow P(a))$.
Since for all $x, y \in [0, 1]$

$$\min\{1, 1 - x + y, 1 - y + x\} = \max\{0, \min\{1 - x + y, 1 - y + x\}\}$$

then, $(x \rightarrow y) \wedge (y \rightarrow x) \equiv (x \rightarrow y) \otimes (y \rightarrow x)$ *and in this case each choice we make between* $\wedge$ *and* $\otimes$ *yields an equivalent result. In particular, if we use the weak conjunction we can immediately see that such a formula belongs to the concave fragment.*

Example 4.5 (Mutually Exclusive Classes)**.** *One can ask, for instance in a collective classification problem, that a certain pattern belongs to one and only one of two (or more) classes whose membership functions are given by two predicates* P, Q*. For short we indicate with* x *and* y *the two grounded predicates corresponding to the class assignation to the same object* a*, namely* $x := P(a)$*,* $y := Q(a)$*. For instance, by means of* $x \vee y$ *we can express that* a *belongs to at least one of the two classes and by* $(x \wedge \neg y) \vee (\neg x \wedge y)$ *that* a *belongs to exactly one class. Such formulas can be translated into* **Ł** *in different ways. However it seems that some choices are more accurate with respect to the initial intent of the formula. For instance, if we translate* $x \vee y$ *with* $x \oplus y$*, then the formula will be satisfied for any pair of* $[0, 1]$*–values* *summing to 1. Therefore in this case, it can be more useful to translate it with the weak disjunction that corresponds to the maximum. For what concerns the exclusive disjunction (xor), it can be fairly represented in Łukasiewicz logic by $(x \otimes \neg y) \vee (\neg x \otimes y)$ that belongs to the convex fragment.*

4.2 Fuzzy Aggregation and Loss Functions

Finally in this section, we consider more general ways to aggregate and enforce the satisfaction of formulas into a learning problem. We already observed (see equation (4.1)) how quantified FOL formulas can be converted into fuzzy logic. In particular, the universal and existential quantifiers can be mapped into the *min* and *max* operations respectively, that are definable in any fuzzy logic. However, a quantifier can be simply seen as a way to aggregate all the possible groundings of a predicate variable that, in turn, are $[0, 1]$–values. Beside the studies on how to aggregate a finite number of real numbers into a single number by generalized convex functions [118], here we are interested in the connection between generated archimedean t-norms and mapping to a certain loss function. In fact, several aggregation functions can be chosen to implement the quantifiers, e.g. we could choose the arithmetic mean for the universal quantifier. Even if this choice may yield some learning benefits, it has no direct justification inside a logic theory. Moreover it does not suggest how to map the functional translation of the formula to a constraint. The satisfaction of a formula φ may be enforced by the constraint expressed in equation (4.2), namely minimizing $1 - f_\varphi$, where f_φ represents the functional conversion of φ. However in principle, one may satisfy φ minimizing a decreasing mapping $d(f_\varphi)$, where $d : [0, 1] \to [0, +\infty]$ with $d(1) = 0$ expressing the cost of violating the formula φ. This is the reason why, we investigated the mapping of formulas into constraints by means of generated t-norm fuzzy logics, and we exploited the same additive generator of the t-norm to map the formula to be satisfied into the functional constraints to be minimized.

4.2.1 Loss Functions by T-Norms Generators

We already introduced a possible way to convert FOL formulas into functional constraints by means of equations (4.1) and (4.2). In particular, this approach allows us to convert Łukasiewicz formulas into convex functional constraints corresponding to the **Ł** negation of such formulas. However, here we explore a different perspective.

Example 4.6. *Given two predicates p, q defined on a same domain $\mathcal{X}$, the role of the quantifiers on some formulas have to be interpreted as follows,*

$$\forall x\, p(x) \vee q(x) \simeq (p(x_1) \vee q(x_1)) \ \mathit{AND} \ldots \mathit{AND} \ (p(x_N) \vee q(x_N))$$

$$\exists x\, p(x) \wedge \neg q(x) \simeq (p(x_1) \wedge \neg q(x_1)) \ \mathit{OR} \ldots \mathit{OR} \ (p(x_N) \wedge \neg q(x_N))$$

where $\mathcal{X} = \{x_1, \ldots, x_N\}$ denotes the set of the available samples for p and q.

Since the universal quantifier can be seen as a general AND, we are interested in converting this AND with a generated archimedean t-norm. For short, we omit the case of the universal quantifier, however it can be managed in a similar way.

Given a certain formula $\varphi(x)$ depending on a variable x that ranges in the set $\mathcal{X}$ and its corresponding functional translation $\Phi(x, \boldsymbol{f}, \boldsymbol{p})$ evaluated on each $x \in \mathcal{X}$, we can consider the conversion of universal quantifiers by means of a t-norm T as well as existential quantifiers by t-conorms. For instance,

$$\forall x\, \varphi(x) \quad \Longrightarrow \quad \Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p}) = g^{-1}\left(\min\{g(0^+), \sum_{x \in \mathcal{X}} g\big(\Phi(x, \boldsymbol{f}, \boldsymbol{p})\big)\}\right) , \tag{4.7}$$

where g is an additive generator of the t-norm T corresponding to the universal quantifier. Since any generator function is decreasing, in order to satisfy $\forall x\, \varphi(x)$ we can enforce the minimization of g applied to $\Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})$ in equation (4.7). Therefore, we get the following term to minimize:

$$\min\{g(0^+), \sum_{x \in \mathcal{X}} g\big(\Phi(x, \boldsymbol{f}, \boldsymbol{p})\big)\} ,$$

and in case the t-norm T is strict, (namely $g(0^+) = +\infty$) this simplifies in the minimization of

$$\sum_{x \in \mathcal{X}} g\big(\Phi(x, \boldsymbol{f}, \boldsymbol{p})\big) . \tag{4.8}$$

For instance, if we take $g(x) = -log(x)$ from (4.8) we get

$$-\sum_{x\in\mathcal{X}} log\big(\Phi(x, \boldsymbol{f}, \boldsymbol{p})\big) \ , \tag{4.9}$$

that corresponds to a generalization to generic fuzzy logic expressions of the cross–entropy loss. We will discuss this choice with more details in Sec. 6.4.

For what concerns the convexity of expressions obtained composing more functions, it is a well-known result that given a concave function h and a convex non-increasing function g defined over a univariate domain, then we have that $f(x) = g(h(x))$ is convex. As a consequence, we get the following result.

Corollary 4.1. *If g is a convex strict generator function and $\Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})$ is concave, then the function in equation (4.8) expressing the constraint corresponding to a certain formula is convex.*

An example of an application case of the above mentioned result is given considering e.g. $g(x) = -log(x)$ or $g(x) = 1-x$ and Φ built from the concave Łukasiewicz framgent (see Definition 4.2).

As we already pointed out in Sec. 2.3, if g is an additive generator for a t-norm T, then the residual implication and the biresidum with respect to T are given by equations (2.9) and (2.10). In particular, if p, q are two predicates functions and assuming T is strict for simplicity, the following universally closed formulas are converted as:

$$\begin{aligned} \forall x\, p(x) &\implies g^{-1}\big(\sum_x g(p(x))\big) \\ \forall x\, p(x) \to q(x) &\implies g^{-1}\big(\sum_x \max(0, g(q(x)) - g(p(x))\big) \\ \forall x\, p(x) \leftrightarrow q(x) &\implies g^{-1}\big(\sum_x |g(p(x)) - g(q(x))|\big) \end{aligned} \tag{4.10}$$

Obviously in case of more complex formulas, some occurrences of the generator may be simplified, as for instance the formula $\forall x\, (p(x) \wedge q(x)) \rightarrow r(x)$

$$\underbrace{g^{-1}\left(\sum_x g\left(\underbrace{g^{-1}\left(\max\left\{0, g(r(x)) - g\left(\underbrace{g^{-1}(g(p(x)) + g(q(x)))}_{conjunction}\right)\right\}\right)}_{implication}\right)\right)}_{quantifier} ,$$

can be reduced to the simpler expression:

$$g^{-1}\left(\sum_x \max\{0, g(r(x)) - g(p(x)) - g(q(x))\}\right) .$$

Further, one may think to consider customized loss components that are more suitable for a certain learning problem or exploiting the described construction to get already known machine learning loss, as in equation (4.9).

Example 4.7. *For instance, if $g(x) = \frac{1}{x} - 1$ with corresponding t-norm $T(x, y) = \frac{xy}{x+y-xy}$, the functional constraint (4.8) that is obtained applying g to the formula $\forall x\, p(x) \rightarrow q(x)$ is given by*

$$\sum_x \max\left\{0, \frac{1}{q(x)} - \frac{1}{p(x)}\right\} .$$

Example 4.8. *If $g(x) = 1 - x^2$ (note that its generated t-norm is nilpotent in this case, indeed $g(0) = 1$) the corresponding constraint is given by*

$$\min(1, \sum_x \max(0, (p(x))^2 - (q(x))^2)) .$$

In the next chapters, we will see some cases of interest exploiting the theoretical results presented in this chapter. For instance, the definition of the concave and convex fragment of Łukasiewicz logic are exploited in Ch. 5 to extend some classical machine learning schema still preserving convexity. Arbitrary construction of loss functions by means of fuzzy aggregation functions and additive generators are instead exploited in Ch. 6, where logical constraints are mapped by a chosen generator into components of an overall loss function.

Chapter 5

Learning with Convex Logical Constraints

In this chapter we discuss some learning frameworks where we are able to get some theoretical insights when exploiting the convex Łukasiewicz fragment. In the big picture, we are interested in the learning of a set of real-valued functions $\mathbf{P} = \{p_j^{(a_j)} : \mathbb{R}^{n_j} \to \mathbb{R},\, j \in \mathbb{N}_J$[1]$,\, n_j, a_j > 0\}$, denoting logical predicates, provided with some factual (supervisions) and abstract knowledge (logical formulas) on them. Throughout this chapter, each objective function $p_j^{(a_j)}$ is supposed to be evaluated on a supervised training set $\mathscr{L}_j$ and an unsupervised training set $\mathscr{U}_j$, where:

$$\mathscr{L}_j = \{(\mathbf{x}_l^j, y_l^j) : \mathbf{x}_l^j \in \mathbb{R}^{n_j}, y_l^j \in \{-1,1\}, l \in \mathbb{N}_{l_j}\}\ ,$$

$$\mathscr{U}_j = \{\mathbf{x}_u^j \in \mathbb{R}^{n_j} : u \in \mathbb{N}_{u_j}\}\ .$$

In addition, we define $\mathscr{S}_j = \mathscr{U}_j \cup \mathscr{L}'_j$ (where $\mathscr{L}'_j = \{\mathbf{x}_l^j : (\mathbf{x}_l^j, y_l^j) \in \mathscr{L}_j\}$) as the set containing the whole sample of points on which the j–th objective is evaluated. For any j, the logical constraints related to the j–th predicate will be enforced on the set $\mathscr{S}_j$. Since any quantifier is applied to a specific argument of a predicate, it can be useful to decompose by components (with respect to the arity a_j) the range of vectors $\mathbf{x}_s^j \in \mathscr{S}_j$, namely we consider $\mathscr{S}_j = \mathscr{S}_{j1} \times \ldots \times \mathscr{S}_{ja_j}$ so that each $\mathscr{S}_{jk}$ is the domain of the k–th argument of the predicate $p_j^{(a_j)}$. In the following, we indicate with s_j the cardinality of $\mathscr{S}_j$, $S = s_1 + \ldots + s_J$ and $U = u_1 + \ldots + u_J$. Further, we suppose we are given a set of Łukasiewicz FOL formulas, $KB = \{\varphi_h : h \in \mathbb{N}_H\}$ whose predicates are functions in $\mathbf{P}$ and without any function symbol, i.e. $\mathbf{F} = \emptyset$. With respect to the notation, in the following we adopt some abbreviations. For instance, we will omit the superscript j on points, the arity on predicates and we write

[1]For short, $\mathbb{N}_n = \{1, 2, \ldots, n\}$ denotes the set of the first $n > 0$ natural numbers.

p_{jl} instead of $p_j(\mathbf{x}_l)$. In addition, we write $\boldsymbol{p} = (p_1, \ldots, p_J) : \mathbb{R}^n \to \mathbb{R}^J$, where $n = n_1 + \ldots + n_J$, for the overall objective function.

5.1 Kernel Machines

Kernel methods are a class of algorithms for pattern analysis that exploit high-dimensional feature representation. These methods are widely studied, also in the particular case of structured data [46] of which logical knowledge may be seen as a special instance. In Sec. 2.4.3 we already introduced the well-known *Support Vector Machines* (SVMs), one of the mostly employed kernel machines algorithm. Basically, SVM is a supervised learning model aiming at separate a set of points that belong to different classes with an as large as possible margin. One of the advantages of SVMs is that learning can be efficiently solved by quadratic optimization algorithms both in the primal and in the dual space. In the following, we show how to extend its classical formulation to include logical constraints still preserving the quadratic programming schema. From now on in this section, we assume that each objective function p_j belongs to some *Reproducing Kernel Hilbert Space* (RKHS) $\mathcal{H}_j$. By means of the reproducing property, as shown in equation (5.1), p_j can be represented as an expansion of the kernel function $k_j \in \mathcal{H}_j$,

$$p_j(x) = \sum_{s=1}^{s_j} \alpha_s^j \cdot k_j(x, x_s) \tag{5.1}$$

where $\alpha_s^j \in \mathbb{R}$ are the model parameters for the learned solution and $x_s \in \mathcal{S}$. We note that in an experimental setting, the choice of the kernel is empirically validated. However, typical choices fall into *linear*, *polynomial* or *gaussian* kernels.

5.1.1 SVMs with Logical Constraints

In the following, according to the learning from constraints paradigm, we start describing the constraints involved in the learning problem. After that, it will be straightforward to formulate the overall loss function to be optimized. As we will see, the choice of the concave fragment to translate the logical formulas into functional constraints will guarantee to deal with convex (even linear) constraints.

Constraints

In the considered learning problem, each learnable task function has to be treated as a predicate subject to both supervisions and the satisfaction of logical knowledge. Hence, the constraints may be divided into the following three categories.

- *Consistency Constraints* derive from the fact that predicate functions have to be evaluated into $[0,1]$ in order to guarantee the consistency of the FOL formula evaluations.
- *Pointwise constraints* enforce the behavior of the task functions on the supervised examples provided in the training set.
- *Logical constraints* establish some abstract relations that must hold between the predicates by means of FOL formulas.

Consistency Constraints The function $\boldsymbol{p}$ represents a tuple of logical predicates. In principle, when we write any p_j according to equation (5.1), we have no guarantees p_j is evaluated in $[0,1]$. In previous works (e.g. [33]) this is achieved applying a sigmoid function to predicates before enforcing the logical constraints. Since in general the sigmoid does not preserve convexity (as well as concavity), we opt for a different strategy. In order to guarantee the consistency in the definition of logical constraints, for every $j \in \mathbb{N}_J$, we add the following hard constraints:

$$0 \leq p_j(\mathbf{x}_s) \leq 1, \quad \text{for every } \mathbf{x}_s \in \mathscr{S}_j \ . \tag{5.2}$$

Pointwise Constraints For any $j \in \mathbb{N}_J$, the supervised set $\mathscr{L}_j$ provides pairs of supervised examples $(\mathbf{x}_l, y_l)$ for p_j. If $y_l = 1$ then $\mathbf{x}_l$ belongs to the true class, so we would like to have $p_j(\mathbf{x}_l) \geq 1$, whereas if $y_l = -1$ then $\mathbf{x}_l$ belongs to the false class and we would like to have $p_j(\mathbf{x}_l) \leq 0$. In order to avoid the feasible set of solutions for the optimization problem to be empty, we allow such constraints to be partially violated. Summing up, for every

$j \in \mathbb{N}_J$, supervisions are enforced requiring the satisfaction of soft constraints according to the classical approach with hinge loss functions:

$$y_l(2p_j(\mathbf{x}_l) - 1) \geq 1 - 2\xi_{jl} \text{ with } \xi_{jl} \geq 0, \quad \text{for every } \mathbf{x}_l \in \mathscr{L}_j \ , \tag{5.3}$$

where the slack variable ξ_{jl} denotes the smallest nonnegative number satisfying the constraint. Pointwise constraints are slightly modified with respect to the usual formulation, since the supervisions are labelled with values $y_l = \pm 1$ whereas the predicates are supposed to assume 0–1 values denoting classical logic truth values.

Logical Constraints Logical constraints arise from the knowledge base KB that is supposed to be a collection of FOL **Ł**–formulas. Without loss of generality for the Łukasiewicz logic case, they may be assumed to be in *prenex normal form.* According to equation (4.3) each quantified formula can be replaced with a propositional one once all the predicates are grounded on their domains of evaluation. We denote the set of such *propositionalized* formulas, where the grounded predicates are considered as $[0, 1]$ propositional variables, by KB'. For what concerns logical constraints, every p_j is supposed to be evaluated on the set $\mathscr{S}_j$. Since the formulas in KB' depend, in general, on all the possible groundings of their occurring predicates, for $j \in \mathbb{N}_J$ it is useful to indicate with $\bar{\boldsymbol{p}}_j$ the vector of all groundings of p_j in $\mathscr{S}_j$, namely $\bar{\boldsymbol{p}}_j = (p_{j1}, \ldots, p_{js_j})$ and $\bar{\boldsymbol{p}} = (\bar{\boldsymbol{p}}_1, \ldots, \bar{\boldsymbol{p}}_J) \in [0, 1]^S$.

Example 5.1. *Let us consider* $\mathbf{P} = \{p_1, p_2\}$ *and* $KB = \{\varphi_1, \varphi_2\}$ *with:*

$$\varphi_1 : \forall x \exists y (p_1(x) \rightarrow p_2(x, y)), \qquad \varphi_2 : \forall x (p_1(x) \vee \neg p_1(x)).$$

Given $\mathscr{S}_{11} = \mathscr{S}_{21} = \{x_1, x_2\}$ *and* $\mathscr{S}_{22} = \{y_1, y_2\}$*, we have* $\mathscr{S}_1 = \mathscr{S}_{11}$ *and* $\mathscr{S}_2 = \{(x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2)\}$*, hence the grounding vectors for the two predicates are*

$$\bar{\boldsymbol{p}}_1 = (p_{11}, p_{12}) \ , \quad \bar{\boldsymbol{p}}_2 = (p_{21}, p_{22}, p_{23}, p_{24}) \ .$$

Therefore, we get $KB' = \{\varphi'_1, \varphi'_2\}$ *where* φ'_1 *and* φ'_2 *are respectively:*

$$\varphi'_1 : \ [(p_{11} \rightarrow p_{21}) \vee (p_{11} \rightarrow p_{22})] \wedge [(p_{12} \rightarrow p_{23}) \vee (p_{12} \rightarrow p_{24})] \ ,$$
$$\varphi'_2 : \ (p_{11} \vee \neg p_{11}) \wedge (p_{12} \vee \neg p_{12}) \ .$$

Formulas in KB' are now expressed in propositional Łukasiewicz logic **Ł** and so we can use all the results reported in the previous chapter. In particular, every $n--$ary formula φ is isomorphic to a McNaughton function $f_\varphi : [0,1]^n \to [0,1]$. For the sake of simplicity, we indicate by f_h the function corresponding to the propositional formula φ'_h. In addition to make the notation as uniform as possible, although any formula will depend in general on only few predicates, we write $f_h = f_h(\bar{\boldsymbol{p}})$ as, in general, they depend on the grounding vector of all the predicates.

With the introduced notation, we are now ready to express the logical constraints exploiting equation (4.2)[2]. Also in this case, a new slack variable is introduced for each formula to allow the soft violation of the constraints. Hence, for every $h \in \mathbb{N}_H$, we enforce the soft satisfaction of:

$$1 - f_h(\bar{\boldsymbol{p}}) \leq \xi_h \text{ with } \xi_h \geq 0 \ . \tag{5.4}$$

If $KB' \subseteq (\wedge, \oplus)^*$, namely if the formulas are built from the concave fragment, then f_h is a concave function and $1 - f_h$ is the convex function corresponding to the formula $\neg\varphi_h$. It is worth to notice that the considered McNaughton functions are convex in the space of grounded predicates, however the formulas in KB' basically depend on the parameters which determine each predicate function. Since we suppose that each objective function belongs to a certain RKHS, other than the representation in equation (5.1), we can write

$$p_j(\mathbf{x}) = \omega'_j \cdot \phi_j(\mathbf{x}) + b_j \ , \tag{5.5}$$

where $\phi_j : \mathbb{R}^{n_j} \to \mathbb{R}^{N_j}$ is a feature map determined by the j–th kernel function k_j of $\mathcal{H}_j$, $\omega_j \in \mathbb{R}^{N_j}$ is said the j–th weight vector and $b_j \in \mathbb{R}$ the bias. If we assume to evaluate the predicates only on the known fixed training set and we set $\hat{\omega}_j = (\omega'_j, b_j)'$, then the values of predicates are totally determined by the matrix $\hat{\omega} = (\hat{\omega}_1, \ldots, \hat{\omega}_J)$. This entails that the formulas will be evaluated by composition on the weight space and therefore we need to guarantee the convexity of McNaughton functions on this space. Thanks to the linear form assumed by each objective function in the feature space (in general $N_j >> n_j$ or even N_j may be infinite), the following lemma applies and guarantees convexity of the functional logical constraints in the weight space too.

[2]This corresponds to apply the Łukasiewicz logic generator $g(x) = 1 - x$ to $f_h(\bar{\boldsymbol{p}})$ (that corresponds to a **Ł**–formula), as also proposed in Sec. 4.2.1.

Lemma 5.1. *Let $f : Y \subseteq \mathbb{R}^m \to \mathbb{R}$ be a concave (convex) function and $g : X \subseteq \mathbb{R}^d \to Y$ such that $g(x) = Ax + b$ with $A \in \mathbb{R}^{m,d}, b \in \mathbb{R}^m$. Then the function $h : X \to \mathbb{R}$ defined by $h = f \circ g$ is concave (convex) in X.*

Summing up, if we consider only formulas that belong to the concave fragment we can embed the logical constraints into the overall loss function by means of convex functional constraints. However, since we suppose to deal with Łukasiewicz formulas, any functional constraint is a McNaughton function as well, and in particular a piecewise linear function. Therefore according to equation (4.4), for every $h \in \mathbb{N}_H$, we have :

$$1 - f_h(\bar{\boldsymbol{p}}) = \max_{i=1,\cdots,I_h} \left(M_i^h \cdot \bar{\mathbf{p}} + q_i^h \right) \leq \xi_h$$

if and only if

$$M_i^h \cdot \bar{\boldsymbol{p}} + q_i^h \leq \xi_h \quad \text{for all } i \in \mathbb{N}_{I_h}, \tag{5.6}$$

where $M_i^h \in \mathbb{R}^{1,S}$ and $q_i^h \in \mathbb{R}$ are integer coefficients determined by the structure of the formula $\neg\varphi_h'$. This means that for every $h \in \mathbb{N}_H$, we can consider the I_h logical constraints expressed by (5.6) in place of (5.4). Even if the number of constraints is increased, each of them is now an affine function, so we can deal with a quadratic programming problem. It is worth to notice that, since the distributive law holds in any fragment for the strong connective with respect to the weak one, we can easily rewrite any concave formula as a weak conjunction of strong disjunctions as well as any convex formula as a weak disjunction of strong conjunctions. Thereafter is straightforward to get the integer coefficients of the affine functions.

For instance, given $\varphi \in (\wedge, \oplus)^*$ with $\varphi :\ (x_1 \oplus \neg x_2) \wedge (x_1 \oplus x_2)$, we have:

$$f_\varphi = \min\{1, x_1 - x_2 + 1, x_1 + x_2\} \ .$$

Optimization Problem

In order to learn the objective functions we have to find the optimal values for the weight matrix. As in SVMs, we look for a solution maximizing the margin between the false and the true class while satisfying the constraints. Functional logical constraints are expressed by their linear counterparts (equation (5.6)), such that we can formulate the multi-task learning problem as quadratic optimization both in the primal and in the dual space.

Primal Problem (5.7)

$$\min_{\omega,\xi} \frac{1}{2}\sum_{j\in\mathbb{N}_J} ||\omega_j||^2 + C_1 \sum_{\substack{j\in\mathbb{N}_J\\ l\in\mathbb{N}_{l_j}}} \xi_{j_l} + C_2 \sum_{h\in\mathbb{N}_H} \xi_h \quad \text{subject to:}$$

$$\begin{array}{ll} y_l(2p_j(\mathbf{x}_l)-1) \geq 1 - 2\xi_{j_l}, & \xi_{j_l} \geq 0, \\ M_i^h \cdot \bar{\boldsymbol{p}} + q_i^h \leq \xi_h, & \xi_h \geq 0, \\ 0 \leq p_j(\mathbf{x}_s) \leq 1, & \end{array}$$

where $j \in \mathbb{N}_J$, $l \in \mathbb{N}_{l_j}$, $(\mathbf{x}_l, y_l) \in \mathscr{L}_j$, $h \in \mathbb{N}_H$, $i \in \mathbb{N}_{I_h}$, $s \in \mathbb{N}_{s_j}$, $\mathbf{x}_s \in \mathscr{S}_j$ and C_1, C_2 are positive real parameters to modulate the degree of satisfaction of the constraints and determined by cross validation.

Remark 5.1. *The constants C_1 and C_2 express the (possibly different) degree of satisfaction for the pointwise and logical constraints respectively. It is worth noticing that the supervisions can be seen as atomic logical constraints or their negation. However we decided to keep them separated in this formulation both for clarity with respect to the usual SVM literature and for considering different values for the constants. In principle one can weigh any constraint differently. However in this case, we suppose to have the same degree of belief on all the supervisions as well as on all the logical formulas.*

The primal problem (5.7) can be reformulated as the minimization of a Lagrangian function, obtaining

$$\mathcal{L}(\hat{\omega}, \xi, \lambda, \mu, \eta) = \frac{1}{2}\sum_{j=1}^{J} ||\omega_j||^2 + C_1 \sum_{j=1}^{J}\sum_{l=1}^{l_j} \xi_{j_l} + C_2 \sum_{h=1}^{H} \xi_h - \sum_{j=1}^{J}\sum_{l=1}^{l_j} \mu_{j_l}\xi_{j_l} +$$

$$- \sum_{j=1}^{J}\sum_{l=1}^{l_j} \lambda_{j_l}(y_l(2p_j(\mathbf{x}_l)-1)-1+2\xi_{j_l}) - \sum_{h=1}^{H}\sum_{i=1}^{I_h} \lambda_{h_i}(\xi_h - M_i^h \cdot \bar{\mathbf{p}} - q_i^h) - \sum_{h=1}^{H} \mu_h \xi_h +$$

$$- \sum_{j=1}^{J}\sum_{s=1}^{s_j} \eta_{j_s} p_j(\mathbf{x}_s) - \sum_{j=1}^{J}\sum_{s=1}^{s_j} \bar{\eta}_{j_s}(1 - p_j(\mathbf{x}_s)) \ , \qquad (5.8)$$

with the **KKT**–conditions, for all $j \in \mathbb{N}_J, l \in \mathbb{N}_{l_j}, h \in \mathbb{N}_H, i \in \mathbb{N}_{I_h}, s \in \mathbb{N}_{s_j}$:

$$\begin{gathered}
\xi_{j_l} \geq 0,\ \xi_h \geq 0,\ \lambda_{j_l} \geq 0,\ \mu_{j_l} \geq 0,\ \lambda_{h_i} \geq 0,\ \mu_h \geq 0,\ \eta_{j_s} \geq 0,\ \bar{\eta}_{j_s} \geq 0, \\
\lambda_{j_l}(y_l(2p_j(\mathbf{x}_l) - 1) - 1 + 2\xi_{j_l}) = 0,\ \mu_{j_l}\xi_{j_l} = 0,\ \lambda_{h_i}(\xi_h - M_i^h \cdot \bar{\boldsymbol{p}} - q_i^h) = 0, \\
\mu_h\xi_h = 0,\ \eta_{j_s}p_j(\mathbf{x}_s) = 0,\ \bar{\eta}_{j_s}(1 - p_j(\mathbf{x}_s)) = 0,\ y_l(2p_j(\mathbf{x}_l) - 1) - 1 + 2\xi_{j_l} \geq 0, \\
\xi_h - M_i^h \cdot \bar{\boldsymbol{p}} - q_i^h \geq 0,\ p_j(\mathbf{x}_s) \geq 0,\ 1 - p_j(\mathbf{x}_s) \geq 0.
\end{gathered}$$

The problem can be solved both in the primal and dual space, because convexity guarantees that the KKT–conditions are also sufficient and the duality gap is null. However, to derive the dual space formulation we apply the null gradient condition to the derivatives of $\mathcal{L}$ with respect to every $\omega_j, b_j, \xi_{j_l}, \xi_h$.

$$\nabla_{\omega_j}\mathcal{L} = \omega_j - 2\sum_l \lambda_{j_l} y_l \phi_j(\mathbf{x}_l) + \sum_{h,i} \lambda_{h_i} \sum_u M_{i,u}^h \phi_j(\mathbf{x}_u) - \sum_s (\eta_{j_s} - \bar{\eta}_{j_s})\phi_j(\mathbf{x}_s) = 0\ ;$$

$$\frac{\partial \mathcal{L}}{\partial b_j} = -2\textstyle\sum_l \lambda_{j_l} y_l + \sum_{h,i} \lambda_{h_i} \sum_u M_{i,u}^h - \sum_s (\eta_{j_s} - \bar{\eta}_{j_s}) = 0\ ;$$

$$\frac{\partial \mathcal{L}}{\partial \xi_{j_l}} = C_1 - 2\lambda_{j_l} - \mu_{j_l} = 0\ ;$$

$$\frac{\partial \mathcal{L}}{\partial \xi_h} = C_2 - \textstyle\sum_i \lambda_{h_i} - \mu_h = 0\ .$$

Hence if we substitute, we get:

$$\begin{gathered}
\theta(\lambda, \eta) = \mathcal{L}(\hat{\omega}^*, \xi^*, \lambda, \mu, \eta) = -\frac{1}{2}\sum_j [4\sum_{l,l'} \lambda_{j_l}\lambda_{j_{l'}} y_l y_{l'} k_j(\mathbf{x}_l, \mathbf{x}_{l'}) + \\
+ \sum_{\substack{h,i \\ h',i'}} \lambda_{h_i}\lambda_{h'_{i'}} \sum_{u,u'} M_{i,u}^h M_{i',u'}^{h'} k_j(\mathbf{x}_u, \mathbf{x}_{u'}) + \sum_{s,s'} (\eta_{j_s} - \bar{\eta}_{j_s})(\eta_{j_{s'}} - \bar{\eta}_{j_{s'}}) k_j(\mathbf{x}_s, \mathbf{x}_{s'}) + \\
-4\sum_{l,h,i} \lambda_{j_l}\lambda_{h_i} y_l \sum_u M_{i,u}^h k_j(\mathbf{x}_l, \mathbf{x}_u) + 4\sum_{l,s} \lambda_{j_l} y_l (\eta_{j_s} - \bar{\eta}_{j_s}) k_j(\mathbf{x}_l, \mathbf{x}_s) + \\
-2\sum_{h,i,s} \lambda_{h_i}(\eta_{j_s} - \bar{\eta}_{j_s}) \sum_u M_{i,u}^h k_j(\mathbf{x}_u, \mathbf{x}_s)] + \sum_{j,l} \lambda_{j_l} + \sum_{h,i} \lambda_{h_i}(\frac{1}{2}\sum_u M_{i,u}^h + q_i^h) + \\
-\frac{1}{2}\sum_{j,s} (\eta_{j_s} + \bar{\eta}_{j_s}).
\end{gathered}$$

Finally, we can formulate the dual problem as:

Dual Problem (5.9)

$$\max\ \theta(\lambda,\eta) \qquad\qquad \text{subject to}$$

$$\begin{gathered}
\sum_{h=1}^{H}\sum_{i=1}^{I_h}\lambda_{h_i}\sum_{u=1}^{u_j} M^h_{i,u} = 2\sum_{l=1}^{l_j}\lambda_{j_l} y_l + \sum_{s=1}^{s_j}(\eta_{j_s}-\bar{\eta}_{j_s})\\
0 \le \lambda_{j_l} \le C_1\\
0 \le \lambda_{h_i} \le C_2\\
\eta_{j_s} \ge 0,\ \bar{\eta}_{j_s} \ge 0
\end{gathered}$$

where $j \in \mathbb{N}_J$, $l \in \mathbb{N}_{l_j}$, $h \in \mathbb{N}_H$, $i \in \mathbb{N}_{I_h}$, $s \in \mathbb{N}_{s_j}$.

From (5.9) we can find the optimal values $\lambda^*_{j_l}, \lambda^*_{h_i}, \eta^*_{j_s}, \bar{\eta}^*_{j_s}$ for all j, l, h, i, s. Then, for all $j = 1, \ldots, J$, the solution of the j–th objective with respect to the optimal parameters in the dual space can be written as:

$$p_j(\mathbf{x}) = \omega^*_j \cdot \phi_j(\mathbf{x}) + b^*_j = 2\sum_{l=1}^{l_j}\lambda^*_{j_l} y_l k_j(\mathbf{x}_l,\mathbf{x}) +$$

$$-\sum_{h=1}^{H}\sum_{i=1}^{I_h}\lambda^*_{h_i}\sum_{u=1}^{u_j} M^h_{i,u} k_j(\mathbf{x}_u,\mathbf{x}) + \sum_{s=1}^{s_j}(\eta^*_{j_s}-\bar{\eta}^*_{j_s}) k_j(\mathbf{x}_s,\mathbf{x}) + b^*_j\ . \tag{5.10}$$

As we can see, the solution can be rewritten as an expansion of the j–th kernel k_j with respect to the different types of constraints on the corresponding sample points. The first term corresponds to the pointwise constraints, the second one to the logical constraints and the latter to the consistency constraints.

5.1.2 Experimental Results

Here, we report a first experimental evaluation of the proposed method on two datasets. The first one is a toy problem that allows us to enlighten how logical constraints contribute to the solution of a given problem. The second one is based on the Winston benchmark for image classification.

Toy problem

We consider a multi-task problem, where we want to determine three predicate functions p_1, p_2, p_3 defined on $\mathbb{R}^2$ and with supervised examples in the sets

L_1, L_2, L_3, that are defined as:

$$\begin{aligned} L_1 &= \{((0.1, 0.5), -1), ((0.4, 0.4), -1), ((0.3, 0.8), 1), ((0.9, 0.7), 1)\}\ , \\ L_2 &= \{((0.1, 0.3), -1), ((0.6, 0.4), -1), ((0.2, 0.8), 1), ((0.7, 0.6), 1)\}\ , \\ L_3 &= \{((0.4, 0.2), -1), ((0.9, 0.3), -1), ((0.2, 0.6), 1), ((0.5, 0.7), 1)\}\ . \end{aligned}$$

In Figure 5.1, we show the (unique) solution for the standard kernel machine scheme in which we only take into account the pointwise constraints and we set $C_1 = 15$.

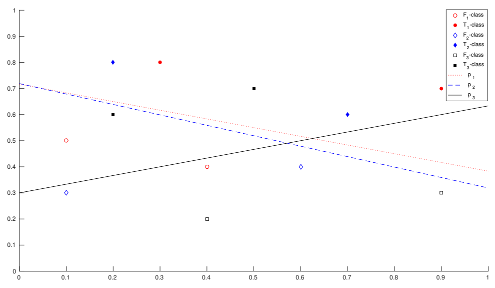

Figure 5.1: *The dotted, dashed and solid lines represent the task functions p_1, p_2 and p_3 respectively. Data points with different shape relate to different predicates and while empty symbols correspond to the false class, the filled ones correspond to the true class.*

Now let us suppose to know, apart from supervisions, some additional relational information about the task functions, expressed by the logic clauses

$$\varphi_1 : \forall x(p_1(x) \to p_2(x)), \qquad \varphi_2 : \forall x(p_2(x) \to p_3(x)),$$

whose intuitive meaning is: whenever a pattern x belongs to the class p_1 then it also belongs to p_2 and similar, if x belongs to p_2 then it has to belong to p_3. To evaluate logical constraints we also exploit a few unsupervised examples. For instance, in Fig. 5.2 we make a comparison between the effect of a single point $P\,(0.8, 0.3)$ (in which logical constraints were violated) and the effect of a larger unsupervised training set

$$U = \{(0.1, 0.5), (0.3, 0.7), (0.5, 0.4), (0.8, 0.3), (0.9, 0.2), (1, 0.5)\}\ .$$

The last plot shows that with just few unsupervised points the boundaries of

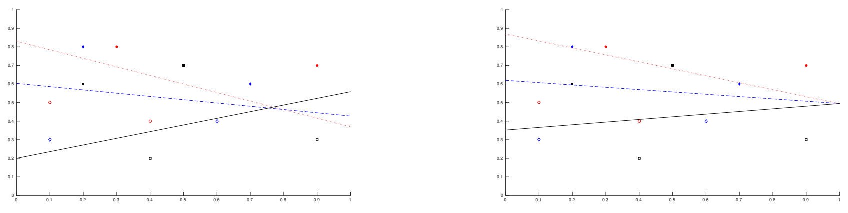

Figure 5.2: *The pictures show, respectively from left to right, the effect of considering as unsupervised examples the single point P and the set U. In both cases we set $C_1 = C_2 = 2.5$.*

the predicates are correctly placed in the (unique) solution in order to satisfy the given logic constraints in the considered domain.

Winston benchmark

The second experimental analysis is based on the animal identification problem originally proposed by P. Winston [151]. This benchmark was initially designed to show the ability of logic programming to guess the animal type from some initial clues.

The dataset is taken from the *ImageNet*[3] database and it consists of 5605 images equally divided into 7 classes, each one representing one animal category: *albatross, cheetah, giraffe, ostrich, penguin, tiger* and *zebra.* The vector of numbers used to represent each image is composed of two parts: one representing the colors in the image, and one representing its shape. In particular, the feature representation contains a 12-dimension of normalized color histogram for each channel in the RGB color space. Furthermore, the SIFT descriptors [93] have been built by sampling a set of images from the dataset and then detecting all the SIFT representations present in at least one of the sampled images. Finally, the SIFT representations have been clustered into 600 *visual words*. The final representation of an image contains 600 values, where the i–th element represents the normalized count of the i–th visual word for the given image (bag-of-descriptors). As previously done in Diligenti et al. [35], the test phase does not get as input a sufficient set of clues to perform classification, but the image representations are used by the

[3]`http://www.image-net.org` .

learning framework to develop the intermediate clues over which inference[4] is performed.

Rules
hair(x) → mammal(x)
milk(x) → mammal(x)
feather(x) → bird(x)
layeggs(x) → bird(x)
mammal(x) ∧ meat(x) → carnivore(x)
mammal(x) ∧ pointedteeth(x) ∧ claws(x) ∧ forwardeyes(x) → carnivore(x)
mammal(x) ∧ hoofs(x) → ungulate(x)
mammal(x) ∧ cud(x) → ungulate(x)
carnivore(x) ∧ tawny(x) ∧ darkspots(x) → cheetah(x)
carnivore(x) ∧ tawny(x) ∧ blackstripes(x) → tiger(x)
ungulate(x) ∧ longlegs(x) ∧ longneck(x) ∧ tawny(x) ∧ darkspots(x) → giraffe(x)
ungulate(x) ∧ white(x) ∧ blackstripes(x) → zebra(x)
bird(x) ∧ longlegs(x) ∧ longneck(x) ∧ black(x) → ostrich(x)
bird(x) ∧ swim(x) ∧ blackwhite(x) → penguin(x)
bird(x) ∧ goodflier(x) → albatross(x)
cheetah(x) ∨ tiger(x) ∨ giraffe(x) ∨ zebra(x) ∨ ostrich(x) ∨ penguin(x) ∨ albatross(x)
mammal(x) ∨ bird(x)
hair(x) ∨ feather(x)
darkspots(x) → ¬ blackstripes(x)
blackstripes(x) → ¬ darkspots(x)
tawny(x) → ¬ black(x) ∧¬ white(x)
black(x) → ¬ tawny(x) ∧¬ white(x)
white(x) → ¬ black(x) ∧¬ tawny(x)
black(x) → ¬ white(x)
black(x) → ¬ tawny(x)
white(x) → ¬ black(x)
white(x) → ¬ tawny(x)
tawny(x) → ¬ white(x)
tawny(x) → ¬ black(x)

Table 5.1: *The KB used for training the models in the Winston image classification benchmark.*

The images have been split into two initial sets: the first one is composed of 2100 images utilized for building the visual vocabulary, while the second set is composed of 3505 images used in the learning process. The experimental analysis has been carried out by randomly sampling from the overall set of the supervisions the labels to keep as training, validation and test set, randomly sampling 50%, 25%, 25% of the supervisions, respectively.

The knowledge about the classification task is expressed by a set of FOL rules. A total of 33 classes are referenced by the KB, the final 7 plus other intermediate classes, each of which is either representing a subset of animals in a taxonomy (like the classes *mammal* or *bird*) or indicating some specific feature of the animals (like *hair* or *darkspots*). Table 5.1 shows the set of

[4]The dataset and code to reproduce the results can be downloaded from `https://github.com/diligmic/ECML2017\_1` .

rules used in this task. The first 15 rules are the same as stated in the original problem definition by Winston. The fact that each image shows one and only one animal classification is expressed by another rule stating that each pattern should belong to only one of the classes representing an animal. Another set of rules forces the semantic consistency among the intermediate classes.

All the images are available at training time, but only the training sample of the labels is made available during training. For each reported experiment, one Kernel Machine is trained for each of the 33 predicates in the KB. Gaussian kernels have been selected to be used in the experiments since they were clearly outperforming both the linear kernel and all the tested variations of the polynomial kernel. In Table 5.2 the summary of the results with respect to different t-norm conversions of the considered logical formulas is reported. Learning with the logic knowledge improved significantly the results. The convex Łukasiewicz fragment yields the highest F1 metric on this benchmark.

	Baseline	Łukasiewicz	Gödel	Product	Convex Łukasiewicz
F1	0.45	0.53	0.52	0.55	**0.56**

Table 5.2: *F1 values for the baseline (no logic knowledge) and using the KB integrated into the learning process using different t-norms.*

5.2 Support Logical Constraints

In Sec. 5.1 we presented a framework extending classical support vector machines (SVMs) with logical constraints still preserving quadratic optimization. A main property of SVMs is that only a small portion of training data is significant to determine the maximum margin separating hyperplane in the feature space, the so called *support vectors*. In this section, we show how this notion can be extended in case of logical constraint. If we approach the problem in the framework of constrained optimization, these vectors will correspond to the *active* constraints in the Lagrangian formulation (see equation (5.8)). This means that we can split the training examples into two categories, the *support vectors*, that completely determine the optimal solution of the problem, and the *straw vectors*. By solving the Lagrangian dual of the optimization problem (see equation (5.10)), the support vectors are those supervised examples corresponding to constraints whose Lagrangian multiplier is not null.

In a similar way, in the general schema of learning from constraints, where possibly several constraints are considered, some of them may turn out to be unnecessary with respect to the learning optimization. Hence, here we aim at extending the definition of support vector to *support constraint* and we provide some criteria to determine which constraints can be removed from the learning problem still yielding the same optimal solutions. Moreover, in the particular case of logical constraints, we are able to enlighten deepen connections between *unnecessary constraints* and logical deduction.

The notion of *support constraints* has already been considered in [58, 59] to provide an extension of the concept of support vector when dealing with learning from constraints. The idea is based on the definition of entailment relations among the constraints and the possibility of constraint checking on data distribution. In this section, we provide a formal definition of *unnecessary constraints* that refines the concept of support constraint and we provide some theoretical results that characterize the presence of such constraints. These results are illustrated by examples that show in practice how the conditions are verified. The main idea is that unnecessary constraints can be removed from a learning problem without modifying the set of optimal solutions. In a similar spirit, with the specific goal to define algorithms accelerating the search for solutions in optimization problems, it is worth to mention some related works in the Constraint Reduction (CR) field [71, 142]. In particular, in [72] the authors show how to reduce the computational burden in a convex optimization problem by considering at each iteration the subset of the constraints that contains only the most critical (or necessary) ones. In this sense, our approach allows us to determine theoretically which are the unnecessary constraints as well as to enlighten their logical relations with the other constraints.

5.2.1 Active Constraints

The expression of the optimal solution in equation (5.10) suggests to analyze separately the contribution of the three categories of constraints. For what concerns the first component of the solution, as for the classic soft-margin SVM case [20], the support vectors correspond to the pointwise constraints that are active, namely such that $1 - y_l(2p_j(x_l) - 1) = 2\xi_{jl}$ and $\lambda_{jl} \neq 0$. In the same way, the h–th logical constraint is a support constraint if at least one of its affine components is active, namely if $\lambda_{h_i} \neq 0$ for some $i \leq I_h$. The latter component, corresponding to the hard consistency constraints, yields support vectors only for those points where the learnable functions assume the crisp values $0, 1$. On the opposite, we refer to constraints whose multipliers are null as *straw constraints.*

As we already pointed out, the optimal solution of *Problem (5.7)* is totally determined by the support constraints, however the solution may be not unique in general. In fact, the problem is convex if the Gram matrix of the chosen kernels on the sample is positive-semidefinite (no uniqueness) and strictly-convex if it is positive-definite (unique solution). For both cases, the Lagrangian function associated to the problem may have different multiplier vectors yielding a specific optimal solution. In particular, for a certain optimal solution of *Problem (5.7)*, we can have two different multiplier vectors with different support and straw constraints associated, e.g. see Example 5.3. The purpose of this study is to investigate the effective role of the constraints involved in a multi-task learning problem with logical constraints. In particular, we are interested in constraints that are not necessary for the optimization, even if they may turn out to be supporting for a certain solution. The main results of this section establish some criteria to discover unnecessary constraints and their relationship with the underling Łukasiewicz logic.

About Logical Constraint Multipliers

By construction, pointwise and consistency constraints are both related to a single pattern for a given predicate. This means that the (possible) contribution of these active constraints to the solution in any point, as expressed by equation (5.10), cannot be redistributed to other constraints of the same type. On the opposite, each logical constraint involves in general more predicates eventually evaluated on different points. Hence we may wonder if it exists an opportune vector of Lagrange multipliers yielding the same contribution to the solution for each point, where as much as possible multipliers for the logical constraints are set equal to zero.

At first, let us express the term for logical constraints in equation (5.10) with respect to any training point x_s as

$$\left(\sum_{h=1}^{H} \sum_{i \in I_h^s} \lambda_{h_i}^* m_{s,h_i}^j \right) k_j(x_s, x) \ , \tag{5.11}$$

where I_h^s contains the indexes of the affine functions involving the s–th grounding of the j–th predicate. Since the expression (5.11) corresponds to the overall contribution of the logical constraints to the j–th optimal solution in its s–th point, we are interested in the case that a single constraint contribution can be *assimilated* by the other constraints in the same point for every predicate. Namely, if there exists $\bar{h} \leq H$ such that for every $j \leq J$ and for every $s \leq s_j$, there exist $\bar{\lambda}_{h_i}$ such that:

$$\sum_{i \in I_{\bar{h}}^s} \lambda_{\bar{h}_i}^* m_{s,\bar{h}_i}^j = \sum_{\substack{h=1 \\ h \neq \bar{h}}}^{H} \sum_{i \in I_h^s} \bar{\lambda}_{h_i} m_{s,h_i}^j \ .$$

This issue is addressed initially determining the solutions of *Problem 1* and then looking for a solution with null components for the $\bar{h}$–th constraint. It is also worth to notice that thanks to (5.11), equation (5.10) can be rewritten more compactly as

$$\sum_{s=1}^{s_j} \left(\alpha_{j,s}^{(P)} + \alpha_{j,s}^{(L)} + \alpha_{j,s}^{(C)} \right) k_j(x_s, x) = \sum_{s=1}^{s_j} \alpha_{j,s}^* k_j(x_s, x) \tag{5.12}$$

where $\alpha_j^{(P)}(\lambda_{jl}^*), \alpha_j^{(L)}(\lambda_{h_i}^*), \alpha_j^{(C)}(\eta_{j_s}^*, \bar{\eta}_{j_s}^*)$ denote the vectors of optimal coefficients (depending on optimal Lagrange multipliers) of the kernel expansion for pointwise, logical and consistency constraints respectively.

Given an optimal solution for *Problem (5.7)*, the solutions of *Problem 1* correspond to multiplier values yielding the same optimal predicate functions. In particular, we are interested in finding (if it exists) a solution for *Problem 1* where the multiplier values are null for some constraint.

Problem 1. *Given a solution α for Problem (5.7), find $\lambda \in \mathbb{R}^N$ such that:*

$$M \cdot \lambda = \alpha,$$

where $N = \sum_{h=1}^{H} I_h$, M is the matrix of the m^j_{s,h_i} with $S = \sum_{j=1}^{J} s_j$ rows and N columns and $\alpha = M \cdot (\lambda^*_{h_i})_{h\leq H, i\leq I_h}$ the vector of $\alpha_j^{(L)}$ for $j = 1, \ldots, J$.

Solution 1. To solve *Problem 1*, we find an orthonormal base $v_1, \ldots, v_n$ of $Ker(M) = \{\lambda : M \cdot \lambda = 0\}$, so that any solution can be expressed as:

$$\lambda = \lambda^* + \sum_{i=1}^{n} t_i v_i \ ,$$

for some $t_i \in \mathbb{R}$. Finally, we have the following cases:

(i) if $dim(Ker(M)) = 0$ then the system allows the unique solution λ^*;

(ii) if $dim(Ker(M)) \neq 0$ then infinite solutions exist.

In the first case, the only constraints whose multipliers give null contribution to the optimal solution are the original straw constraints. Whereas in the second case, we look for a solution $\bar{\lambda}$ (if there exists) where $\bar{\lambda}_{\bar{h}_i} = 0$ for any $i \leq I_{\bar{h}}$ for some $\bar{h} \leq H$. Indeed in such a case, we can replace λ^* with $\bar{\lambda}$ subdividing any contribution of the $\bar{h}$–th constraint to the other constraints still obtaining the same optimal solution for the predicates. This is carried out by solving the linear system with $I_{\bar{h}}$ equations $\bar{\lambda}_{\bar{h}_i} = 0$ and n variables $t_1, \ldots, t_n$.

In the following, we will say that a vector $(\lambda_{h_i})_{h\leq H, i\leq I_h}$ is a *solution of Problem 1 with respect to $\bar{h}$*, if it is a solution and $\lambda_{\bar{h}_i} = 0$ for every $i \leq I_{\bar{h}}$.

The Transitive Example

Here we illustrate, by means of some cases solved in MATLAB with the interior-point-convex algorithm, how the method works and we discuss the results to clarify what described so far. In particular, we exploit the *transitive law* as a running example to enlighten how the theoretical results apply.

Example 5.2. *Learn the predicates* p_1, p_2, p_3 *subject to* $\forall x\, p_1(x) \to p_2(x)$, $\forall x\, p_2(x) \to p_3(x)$, $\forall x\, p_1(x) \to p_3(x)$. *Given a common evaluation dataset* $\mathscr{S}$*, the logical formulas can be translated, according to Table 4.1 and equation (5.4) into the following functional to be enforced, to evaluate as less or equal than 0,*

$$\max_{x\in\mathscr{S}}\{0, p_1(x) - p_2(x)\}, \quad \max_{x\in\mathscr{S}}\{0, p_2(x) - p_3(x)\}, \quad \max_{x\in\mathscr{S}}\{0, p_1(x) - p_3(x)\},$$

yielding the following terms for the Lagrangian function in equation (5.8),

$$\lambda_{1_1}(p_1(x_1) - p_2(x_1)), \ldots, \lambda_{3_s}(p_1(x_s) - p_3(x_s)) \ .$$

At first we solve the optimization problem where, to avoid trivial solutions, we provide a few supervisions for the predicates and we exploit a polynomial kernel. To keep things clear, we consider only two points, $\mathscr{S} = \{(1, 0.5), (0.4, 0.3)\}$. *Hence, given the (unique) solution* $\alpha(\lambda^*)$ *of* Problem 5.7 *(see Fig. 5.3), where* $\lambda^* = (0.5549, 0, 0, 0.5706, 0, 0)$, *we have*

$$M_2 = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 \end{pmatrix}$$

$$\textit{and} \quad \alpha = M_2 \cdot \lambda^* = \begin{pmatrix} \lambda^*_{1_1} + \lambda^*_{3_1} \\ \lambda^*_{1_2} + \lambda^*_{3_2} \\ -\lambda^*_{1_1} + \lambda^*_{2_1} \\ -\lambda^*_{1_2} + \lambda^*_{2_2} \\ -\lambda^*_{2_1} - \lambda^*_{3_1} \\ -\lambda^*_{2_2} - \lambda^*_{3_2} \end{pmatrix} = \begin{pmatrix} 0.5549 \\ 0 \\ -0.5549 \\ 0.5706 \\ 0 \\ -0.5706 \end{pmatrix} .$$

In this case all the solutions of Problem 1 are given for any $t_1, t_2 \in \mathbb{R}$ *by*

$$\lambda = \lambda^* + t_1 \cdot \begin{pmatrix} -1 \\ 0 \\ -1 \\ 0 \\ 1 \\ 0 \end{pmatrix} + t_2 \cdot \begin{pmatrix} 0 \\ -1 \\ 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} =$$

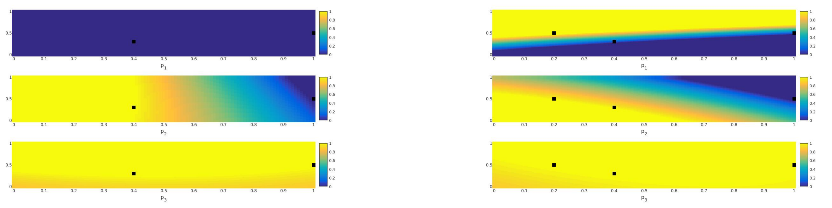

Figure 5.3: *From left to right we report the evaluation of the learned functions p_1, p_2, p_3 in the example space for Example 5.2 and Example 5.3, respectively. Filled squares correspond to the provided sample points.*

$$= \begin{pmatrix} \lambda^*_{1_1} - t_1 \\ \lambda^*_{1_2} - t_2 \\ \lambda^*_{2_1} - t_1 \\ \lambda^*_{2_2} - t_2 \\ \lambda^*_{3_1} + t_1 \\ \lambda^*_{3_2} + t_2 \end{pmatrix} = \begin{pmatrix} 0.5549 - t_1 \\ -t_2 \\ -t_1 \\ 0.5706 - t_2 \\ t_1 \\ t_2 \end{pmatrix} ,$$

where the vectors $v_1 = (-1, 0, -1, 0, 1, 0)'$, $v_2 = (0, -1, 0, -1, 0, 1)'$ form a base for $Ker(M_2)$. From this expression, we get that the only way to obtain the same α nullifying the contribution of the third constraint is taking $t_1 = t_2 = 0$, namely taking $\lambda = \lambda^*$. We can also nullify the contribution of the first or of the second constraint taking $t_1 = 0.5549, t_2 = 0$ or $t_1 = 0, t_2 = 0.5706$ respectively. In these cases we get $\lambda^*_1 = (0, 0, -0.5549, 0.5706, 0.5549, 0)'$ and $\lambda^*_2 = (0.5549, -0.5706, 0, 0, 0, 0.5706)'$, but in any case the third one becomes a support constraint.

As we can see from the previous example, the solutions of *Problem 1* are not necessarily vectors of KKT multipliers. Indeed, KKT–conditions include *stationarity, primal feasibility, dual feasibility* and *complementary slackness.* However, if we suppose that every null component in the initial optimal configuration is preserved, then it is enough to ask for non-negativity of all the other components. We call such solutions of *Problem 1* as *KKT–solution.* For instance, in Example 5.2, λ^* is a KKT–solution while λ^*_1 and λ^*_2 are not.

It is worth noticing that we can attempt to find a configuration of multipliers solving *Problem 1* and maximizing the number of logical constraints whose contributions can be distributed to a few logical constraints remaining.

However in general, it is not guaranteed to get a KKT–solution, hence it could be not achievable by solving directly *Problem (5.7)*.

Now lets come back to Example 5.2. Although it is easy to see that the third constraint is deducible from the others with a logic argument, *Problem (5.7)* may give a different perspective in terms of support constraints.

Example 5.3. *Let us consider the same problem as defined in Example 5.2 with the additional point* $(0.2, 0.5)$ *in* $\mathscr{S}$. *In this case we get the solution* $\lambda^* = (0.3520, 0.3453, 0, 1.1529, 0, 0.5631, 0.4202, 0, 0)'$, *hence the third constraint turns out to be initially a support constraint. However we may wonder if there is another solution of Problem 1 where the components of the third constraint are null. The matrix of constraint coefficients* M_2 *is augmented by three rows and three columns corresponding to the additional grounding of the predicates and to the new affine components for the logical constraints on the new point. However, since the logical formulas to be enforced are the same, the coefficient matrix* M_3 *remains quite similar:*

$$M_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 \end{pmatrix}$$

In this case, the dimension of $Ker(M_3)$ *is increased exactly by one, as the number of affine components of any involved logical constraint. This means, we can try to find a* $\bar{\lambda}^*$ *in which a certain constraint has null values. For instance,* $\bar{\lambda}^* = (0.7722, 0.3453, 0, 1.5731,\ 0, 0.5631, 0, 0, 0)$ *is a solution of* Problem 1 *with respect to the third constraint. However, as before it is the only KKT–solution allowing us to remove the contribution of a constraint.*

The matrices associated to this problem for different size samples have another remarkable property due to the involved constraints. Indeed, if we sort the rows and columns grounding-by-grounding, we can write

$$M_2 = \left(\begin{array}{c|c} M_1 & 0 \\ \hline 0 & M_1 \end{array}\right),\ M_3 = \left(\begin{array}{c|c} M_2 & 0 \\ \hline 0 & M_1 \end{array}\right),$$

$$\text{where } M_1 = \begin{pmatrix} 1 & 0 & 1 \\ -1 & 1 & 0 \\ 0 & -1 & -1 \end{pmatrix}$$

is the matrix of *Problem 1* (in case the sample set consists of just one point). As we can expect, $Dim(Ker(M_1)) = 1$ and we can find a KKT–solution with the third component equal to zero. However, we get a more general result. Indeed a base of $Ker(M_1)$ is given by $v = (-1, -1, 1)'$ and any solution of *Problem 1* can be written for some $t \in \mathbb{R}$ as:

$$\lambda = \lambda^* + tv = \begin{pmatrix} \lambda_1^* - t \\ \lambda_2^* - t \\ \lambda_3^* + t \end{pmatrix}. \tag{5.13}$$

For $t = -\lambda_3^*$ we get a solution of *Problem 1* with respect to the third constraint irrespective of the optimal solution determining the vector λ^*. In addition this is a KKT–solution since $\lambda_i^* \geq 0$ for $i = 1, 2, 3$ and $t \leq 0$, whereas the same is not generally true with respect to the other two constraints. Since for this problem, given n points in the sample, M_n can be written as n diagonal blocks of M_1, any solution of *Problem 1* with respect to M_n can be written for some $t_1, \ldots, t_n \in \mathbb{R}$, as

$$\lambda = \begin{pmatrix} \lambda_{1_1}^* - t_1 \\ \lambda_{2_1}^* - t_1 \\ \lambda_{3_1}^* + t_1 \\ \vdots \\ \lambda_{3_2}^* + t_2 \\ \vdots \\ \lambda_{3_n}^* + t_n \end{pmatrix}$$

Taking $t_i = \lambda_{3_i}^*$ for $i = 1, \ldots, n$ we get a KKT–solution for this problem. Therefore we can conclude that given any number of sample points for these constraints we can always find an optimal solution where the third one is a straw constraint, namely where it is unnecessary.

By means of the transitive example we showed some properties of interest under investigation. In the next section, we formalize the notion of *unnecessary* constraint for a learning problem and we discuss some logic and algebraic criteria to discover if a certain constraint is necessary for the optimization.

5.2.2 Unnecessary Constraints

Roughly speaking, we say that a constraint is *unnecessary* for a certain optimization problem if its enforcement does not affect the set of optimal solutions of the problem. The main idea is that if we consider two problems (defined on the same sample and with the same loss), one with and one without the considered constraint, we have the same optimal solutions.

So far, we considered the case of logical constraints only. However, to show how general this approach is, it is worth noticing that, given a predicate p, both pointwise (equation (5.3)) and consistency constraints (equation (5.2)) can be expressed by opportune logical formulas, as well.

- Given a supervised pair (x_l, y_l) for p, we can consider the formulas:

$$\begin{aligned} 1 \rightarrow p(x_l) \quad &\text{if } y_l = 1 \ , \\ p(x_l) \rightarrow 0 \quad &\text{if } y_l = 0 \ . \end{aligned}$$

- The predicate evaluations may be limited to belong to $[0, 1]$ if, for any sample point x_s for p, we consider the formula:

$$(0 \rightarrow p(x_l)) \wedge (p(x_l) \rightarrow 1) \ .$$

In this uniform view, *Problem 1* applies to all the constraints in *Problem (5.7)*.

In the following, we define the notion of unnecessary constraint in both cases of soft and hard constrains. The reason why we have to consider them separately relies on the fact slack variables may affect the logical consequence.

Definition 5.1 (Unnecessary Soft-Constraint)**.** *Let us consider the learnable functions in a set* $\mathbf{P}$ *evaluated on a sample* $\mathscr{S}$ *and* $KB = \{\varphi_1, \ldots, \varphi_H\}$*. We* *say that $\varphi_{\bar{h}} \in KB$ is* unnecessary *for* **SP** *if the following sets, also called α-solution of the problems for short, coincide, namely if*

$$\{\alpha : (\alpha, \xi) \textit{ is a solution of } (\mathbf{SP})\} = \{\alpha : (\alpha, \xi) \textit{ is a solution of } (\overline{\mathbf{SP}})\} \ ,$$

where the two problems are defined as

$$(\mathbf{SP}) \ \min_{\alpha,\xi} L_1(\alpha, \xi), \ \textit{with } 1 - f_h(\bar{\boldsymbol{p}}) \leq \xi_h, \ \xi_h \geq 0, \ \textit{for } h \leq H \ ,$$

$$(\overline{\mathbf{SP}}) \ \min_{\alpha,\xi} L_2(\alpha, \xi), \ \textit{with } 1 - f_h(\bar{\boldsymbol{p}}) \leq \xi_h, \ \xi_h \geq 0, \ \textit{for } h \leq H, h \neq \bar{h} \ ,$$

with $L_1(\alpha, \xi) = L(\alpha) + \sum_{h=1}^{H} \xi_h, \ L_2(\alpha, \xi) = L(\alpha) + \sum_{\substack{h=1 \\ h \neq \bar{h}}}^{H} \xi_h, \ L(\alpha) = \sum_{j \leq J} \alpha_j' K_j \alpha_j.$

As we can observe in Definition 5.1, the loss functions L_1 and L_2 differ for the additive term $\xi_{\bar{h}}$. However a constraint is said unnecessary if the two problems have the same α–solutions that, once provided, determine both the task functions in **P** and the slack variables for the constraints.

The relation between logical inference and deducible constraints arises naturally in this frame. However in general, logical deductive systems involve truth preserving inference, hence to investigate logical criteria we restrict the attention to the hard-constraint case, removing any slack variable. In this way, satisfying a logical constraint means to evaluate the corresponding formula exactly as one. Note that in this case, the constraints may turn out to be unfeasible for a given kernel.

Definition 5.2 (Unnecessary Hard-Constraint)**.** *Let us consider the learnable functions in a set* **P** *evaluated on a sample $\mathscr{S}$ and $KB = \{\varphi_1, \ldots, \varphi_H\}$. We say that $\varphi_{\bar{h}} \in KB$ is* unnecessary *for* **HP** *if the optimal solutions of problems* **HP** *and* $\overline{\mathbf{HP}}$ *coincide, where*

$$(\mathbf{HP}) \ \min_{\alpha} L(\alpha), \ \textit{with } 1 - f_h(\bar{\boldsymbol{p}}) \leq 0, \ \textit{for } h \leq H \ ,$$

$$(\overline{\mathbf{HP}}) \ \min_{\alpha} L(\alpha), \ \textit{with } 1 - f_h(\bar{\boldsymbol{p}}) \leq 0, \ \textit{for } h \leq H, h \neq \bar{h}$$

and $L(\alpha) = \sum_{j \leq J} \alpha_j' K_j \alpha_j.$

It is worth noticing that without slack values the losses of the two problems become the same, but in general the feasible solutions satisfying the constraints can be different. Indeed, if we refer to $\mathcal{F}$ and $\overline{\mathcal{F}}$ for the feasible sets of **HP** and $\overline{\mathbf{HP}}$ respectively, we have in general only $\mathcal{F} \subsetneq \overline{\mathcal{F}}$.

Since all the considered constraints correspond to logical formulas, we can also exploit some consequence relation among formulas in Łukasiewicz logic. In the following, we will write $\Gamma \models \phi$, where $\Gamma \cup \{\phi\}$ is a set of propositional formulas, to express the truth preserving logical consequence in **Ł**, stating that ϕ has to be evaluated as true for any assignment evaluating as true all the formulas in Γ.

Proposition 5.1. *If* $\{\varphi_h : h = 1, \ldots, H, h \neq \bar{h}\} \models \varphi_{\bar{h}}$ *then* $\varphi_{\bar{h}}$ *is unnecessary for* **HP**.

Proof. By hypothesis, any solution satisfying the constraints of $\overline{\mathbf{HP}}$ satisfies the constraints of **HP** as well, namely we have $\mathcal{F} = \overline{\mathcal{F}}$. The conclusion easily follows since the two problems have the same loss function with the same feasible set. □

As it is clear, from the above proposition, it follows that the third constraint in the examples 5.2 and 5.3 is unnecessary. In addition, if we are in presence of more equivalent constraints, we can consider the problem with just one of such constraints instead, still obtaining the same optimal solutions.

One of the main advantages of this approach is providing some criteria to determine the constraints that are not necessary for a learning problem. Indeed, in presence of a large amount of logical rules, *Proposition 5.1* ensures that we can remove all the deducible constraints simplifying the learning process still getting the same solutions. However, the vice versa of *Proposition 5.1* is not achievable, since the logical consequence has to hold for every assignment. The notion of unnecessary constraint is local to a given dataset, indeed the available sample is limited and fixed in general. However, if a constraint is unnecessary then the optimal solutions with or without it coincide and we have that such constraint is satisfied whenever the other ones are satisfied by any optimal assignments. Such consequence among constraints, taking into account only the assignments leading to best solutions on a given dataset, provides an equivalence with the notion of unnecessary constraint. It is interesting to notice that a slightly different version of this consequence has already been considered in [58].

Towards an Algebraic Characterization

In Sec. 5.2.1 we introduced a criterion to discover if a given constraint $\varphi_{\bar{h}}$ can be deactivated solving *Problem 1.* The method consists in finding a vector of Lagrange multipliers with null components corresponding to the groundings of $\varphi_{\bar{h}}$. We are now interested in discovering the relation between this criterion and the notion of unnecessary constraint. Some results are stated by the following propositions.

Proposition 5.2. *If $\varphi_{\bar{h}}$ is unnecessary for* **HP** *then for any optimal solution of* **HP** *there exists a KKT–solution $\bar{\lambda}$ of Problem 1 with respect to $\bar{h}$.*

Proof. If $\varphi_{\bar{h}}$ is unnecessary then **HP** and $\overline{\mathbf{HP}}$ have the same optimal solutions. Let us consider one of them, lets say α^*, where $\alpha^* = \alpha(\lambda^*_{h_i}) = \alpha(\hat{\lambda}^*_{h_i})$ for the two problems with respect to some multipliers vectors $(\lambda^*_{h_i})_{h\leq H,\, i\leq I_h}$ and $(\hat{\lambda}^*_{h_i})_{h\leq H,\, h\neq\hat{h},\, i\leq I_h}$. Since the two vectors of multipliers yield the same optimal solution, then we can define for every $h \leq H, i \leq I_h$ the KKT–solution $\bar{\lambda}$ of *Problem 1* as:

$$\bar{\lambda}_{h_i} = \begin{cases} \hat{\lambda}^*_{h_i} & \text{for } h \neq \bar{h} \\ 0 & \text{otherwise .} \end{cases}$$

□

This has to be thought of as a necessary condition to discover which logical constraints can be removed from **HP** still preserving its optimal solutions. However, the other way round does hold in case either **HP** or $\overline{\mathbf{HP}}$ has a unique solution α^*, but in general we can only prove a weaker result.

Proposition 5.3. *If there exists a KKT–solution $\bar{\lambda}$ of Problem 1 with respect to $\bar{h}$ (for a certain optimal solution $\bar{\alpha}^* = \alpha(\bar{\lambda})$ of* **HP***), then the set of optimal solutions of* **HP** *is included in the set of optimal solutions of* $\overline{\mathbf{HP}}$.

Proof. Given any optimal solution α^* of **HP**, since the problem is (at least) convex, we have $L(\alpha^*) = L(\bar{\alpha}^*)$. At this point, we note that $\bar{\alpha}^*$ is also feasible for $\overline{\mathbf{HP}}$ and that the restriction of $\bar{\lambda}$ on components $h \neq \bar{h}$ is a vector of Lagrange multipliers for $\overline{\mathbf{HP}}$ satisfying the KKT–conditions. The convexity of the problem guarantees that the KKT–conditions are sufficient as well. This means that $\bar{\alpha}^*$ is also an optimal solution for $\overline{\mathbf{HP}}$, hence its loss value is a global minimum and the same holds for α^*. □

In this case we cannot conclude that any optimal solution of $\overline{\mathbf{HP}}$ is an optimal solution for **HP** because in general this solution could be not feasible for this problem. However as we pointed out above, we have the following consequence.

Corollary 5.1. *If* **HP** *(or equivalently* $\overline{\mathbf{HP}}$*) has a unique solution then the assumption of Proposition 5.3 is also sufficient.*

Proof. The solution is unique if the Gram matrix K, that is the same in both the problems, is positive-definite. Hence, requiring the uniqueness of the solution for the two problems is equivalent. Then the claim is trivial from the proof of Proposition 5.3. □

It is worth to notice that since the optimal solutions α^* of *Problem (5.7)* do not depend on the slack variables, we conjecture that the results of this section but Proposition 5.1 also apply to the soft-constraints case. On the opposite, truth preserving logical consequences are no more exploitable if the logical constraints are softly violated.

From Support to Necessary Constraints

The presence of the consistency constraints limits the possible values of the learnable functions between 0 and 1. On the other hand, since we are dealing with hard constraints, the supervisions force the predicates to assume values less than 0 for negative examples and greater than 1 for the positive ones. This means that both pointwise and consisteny constraints will be evaluated exactly to 0 for any optimal solution (if the problem is feasible) on the supervised sample and all the corresponding Lagrange multipliers will be different from zero, namely they will turn out to be support constraints. However, they could be unnecessary constraints for the problem and we could actually remove them from the optimization. By means of the next example we show how the integration with logical constraints can bridge different constraints on the same groundings and admits possibly several unnecessary constraints. Even if quite trivial, the next example enlightens how the method works.

Example 5.4. *Let us consider the same problem as Example 5.2 with only one sample point, i.e.* $\mathscr{S} = \{(0.4, 0.3)\}$ *that is labelled as negative for* p_1 *and positive for both* p_2 *and* p_3*. We express the pointwise constraints as well as the consistency constraints in logical form. All the constraints are obtained requiring the following linear functions to be less or equal than zero:*

$$\begin{array}{ccc}
(\textit{logical}) & (\textit{pointwise}) & (\textit{consistency}) \\
& & \\
p_1(x_1) - p_2(x_1), & p_1(x_1), & -p_1(x_1), p_1(x_1) - 1, \\
p_2(x_1) - p_3(x_1), & 1 - p_2(x_1), & -p_2(x_1), p_2(x_1) - 1, \\
p_1(x_1) - p_3(x_1), & 1 - p_3(x_1), & -p_3(x_1), p_3(x_1) - 1.
\end{array}$$

The coefficients matrix M for such constraints on this point is a rectangular matrix with 3 rows (one per grounded predicate) and 12 columns:

$$\begin{pmatrix}
1 & 0 & 1 & 1 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\
0 & -1 & -1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 1
\end{pmatrix}$$

However in this case the constraints are deterministic. Exploiting the complementary slackness and the condition for the Lagrange multipliers given by Problem 1, we can easily provide several combinations of values for the multipliers yielding the same solution. In this case the Gram matrix K is trivially a positive-definite matrix ($K = 1.25$*) and the solution* $\alpha^* = (0, 0.8, 0.8)$ *provided by a linear kernel is unique. For this simple example we have only two possible KKT–solutions of Problem 1 maximizing the number of unnecessary constraints, namely* $\bar{\lambda} = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0.8, 0, 0.8)'$ *and* $\hat{\lambda} = (0, 0.8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1.6)'$*. This may be easily shown since the complementarity slackness forces* $\lambda_1 = \lambda_3 = \lambda_8 = \lambda_9 = \lambda_{11} = 0$ *and multiplying* *by M the remaining multipliers, it has to be satisfied:*

$$\begin{cases} \lambda_4 - \lambda_7 = 0 \\ \lambda_2 - \lambda_5 + \lambda_{10} = 0.8 \\ -\lambda_2 - \lambda_6 + \lambda_{12} = 0.8 \end{cases}$$

Since **HP** *has a unique solution, from Corollary 5.1, we have two different minimal optimization problems. The first one with only $p_2(x_1) - 1 \leq 0$ and $p_3(x_1) - 1 \leq 0$ and the other one with $p_2(x_1) - p_3(x_1) \leq 0$ and $p_3(x_1) - 1 \leq 0$ as necessary constraints respectively.*

In the end, in this section we proposed some criteria as well as a formal definition to capture the notion of unnecessary constraint in a certain learning from constraint problem. The necessity of a certain constraint is related to the notion of consequences among the other constraints that are enforced at the same time. This is a reason why we decided to deal with logical constraints, that are quite general to include both pointwise and consistency constraints, indeed they provide a very natural way of expressing consequence relations. The logical consequence among formulas is a sufficient condition to conclude that a constraint, corresponding to a certain formula, is unnecessary. However, we also provide an algebraic necessary condition that turns out to be sufficient in case the Gram matrices associated to the employed kernel functions are positive-definite.

5.3 Collective Classification

The logical constraints allow us to express relational information on different objective functions at the same time, hence they can be very suitable for *collective classification* problems [135]. Here, we formulate the collective setting in the same spirit as what we did for kernel machines. The logical formulas are collected in KB and each predicate p_j is supposed to be evaluated on its sample set $\mathscr{S}_j$, we recall that $\bar{\boldsymbol{p}}_j = (p_{j1}, \ldots, p_{js_j})$ denotes the vector of all the possible groundings of p_j and $\bar{\boldsymbol{p}} = (\bar{\boldsymbol{p}}_1, \ldots, \bar{\boldsymbol{p}}_J)$.

In Sec. 5.1, we assumed to learn the objective functions by the training of opportune kernel machines, whereas in this case, we assume that an appropriate model (e.g. a neural network or whatever) has already been trained to compute their values taken as a prior. In the following, we indicate by $\hat{\boldsymbol{p}}_j$ the available vector of priors for the j–th objective function and by $\hat{\boldsymbol{p}}$ their overall concatenation. Even if the priors assume $[0,1]$–values, we have to guarantee the same for the values into the grounding vectors of the objective functions. Then, we enforce again the constraints of equation (5.2) and (5.4), whereas we remove the pointwise constraints (5.3), indeed we only require to be close to the known prior vector.

Given this setting, the considered multi-task learning problem for collective classification may be formulated as follows.

Collective Classification Problem (5.14)

$$\min_{\boldsymbol{p}} \sum_{j\in\mathbb{N}_J} ||\bar{\boldsymbol{p}}_j - \hat{\boldsymbol{p}}_j||^2 + C_1 \sum_{h\in\mathbb{N}_H} \xi_h \quad \text{subject to:}$$

$$\begin{aligned} &1 - f_h(\bar{\boldsymbol{p}}) \leq \xi_h, \quad \xi_h \geq 0,\\ &0 \leq p_j(\mathbf{x}_s) \leq 1, \end{aligned}$$

where $h \in \mathbb{N}_H$, $j \in \mathbb{N}_J$, $s \in \mathbb{N}_{s_j}$, $\mathbf{x}_s \in \mathscr{S}_j$ and C_1 is used to weigh the degree of satisfaction of the logical constraints. This optimization problem aims at finding the grounding for the predicates closest to the given priors and yielding the minimal violation of the constraints. If we assume to deal with formulas in the concave fragment, then the logical constraints can be rewritten according to (5.6) and we still obtain a quadratic programming problem that can be efficiently solved.

Manifold Regularization

As an example, we discuss here the already mentioned logical rule for manifold regularization (4.6). In principle, given a binary predicate $R(x, y)$ and a unary predicate P defined on the same domain, we can require that P assumes as close as possible values on those points that are related by R. For instance, the topological properties of the original domains of the predicates are not explicitly represented in the considered setting, apart from the values assigned for the given priors. This suggests to consider a predefined binary relation $R(x, y)$ expressing the membership of the two points to a same manifold. For instance, a spatial regularization manifold can be defined as

$$R(x_1, x_2) = \exp\left(-\frac{||x_1 - x_2||^2}{\sigma^2}\right) ,$$

where σ is a neighbourhood width parameter. In order to apply the manifold regularization for a certain predicate p_j with respect to a certain binary predicate R, we can enforce the satisfaction of the following logical formula, where for each $x_1, x_2 \in \mathscr{S}_j$, $R_{12} = R(x_1, x_2)$, $p_{j1} = p_j(x_1)$, $p_{j2} = p_j(x_2)$:

$$R_{12} \rightarrow ((p_{j1} \rightarrow p_{j2}) \wedge (p_{j2} \rightarrow p_{j1})) ,$$

that yields the convex constraint

$$\max\{0,\, R_{12} + p_{j1} - p_{j2} - 1,\, R_{12} - p_{j1} + p_{j2} - 1\} \leq \xi ,$$

and also to the following linear constraints

$$\begin{aligned} R_{12} + p_{j1} - p_{j2} - 1 \leq \xi \; , \\ R_{12} - p_{j1} + p_{j2} - 1 \leq \xi \; . \end{aligned}$$

5.3.1 Experimental Results

For the evaluation of a collective classification setting, we consider once again the image classification task originally proposed by P.Winston [151]. Since we already described the problem in Sec. 5.1.2, we only mention the differences on the considered dataset and those due to the exploited models. The dataset used in this section is composed by 3505 images with size 32×32 pixels, occurring at different distances, angles and poses. In this case, we assume to be in a transductive context, namely all the images are available at training time but only a subset of the supervisions are available. In particular, in the experiments the amount of training supervisions is varied between 10% and 90%. The knowledge domain is expressed in terms of first–order logic rules as shown in Table 5.1, where 7 of the predicates in the KB correspond to the final animal classes, and the others are intermediate predicates that help in determining the final classes during the inference process performed by collective classification.

A feedforward neural network having one single output neuron and a single hidden layer containing 30 neurons was trained for each final or intermediate

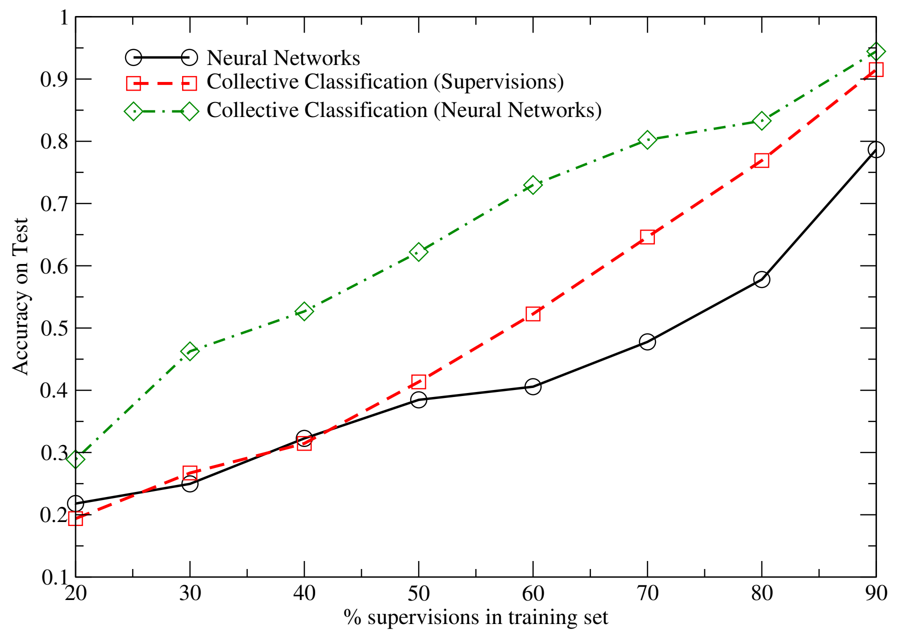

Figure 5.4: *Accuracy of the classifiers varying the amount of available training supervisions.*

predicate in the KB. The single output neuron used a sigmoidal activation function, while the hidden neurons used a rectified linear activation function. The networks are trained against the training set labels using a quadratic cost function on the output. Resilient backpropagation [105, 121] was used to accelerate the convergence of the training process, which was executed for 500 full-batch iterations and using 0.0001 as initial learning rate for all the weights. The network outputs are used to initialize the values of the grounded predicates for the subsequent collective classification step.

Three settings are compared in the experiments. In the first setting, the neural networks classify directly the images based on their input feature based representation without any logic knowledge. In the second, the proposed collective classification, based on the given KB, is applied after initializing the grounded predicate values using only the supervisions available in the train set, whereas setting the other values to 0.5. Finally, in the third setting, the collective classification is performed after the initialization of each grounded predicate with the output of the corresponding neural network for that grounding. Basically, this last classifier in general provides a better prior also for those nodes that are not supervised. In Figure 5.4 we report the accuracy on the test labels for different percentages of the available supervisions for training. The accuracy is computed on the 7 exclusive final animal classes, where the class assignment for a pattern is performed via an argmax of the output classification values.

Collective classification exploiting the KB significantly improves the performances of the neural network classifiers. In particular, the inference step consolidates the final assignments by fixing the inconsistencies of the classifier outputs using the prior knowledge. Among the two collective classification settings, the best performance is obtained by exploiting the outputs of the neural networks as priors. This is because the neural networks not only fit well the available supervisions, but also provide a better prior for the unsupervised grounded predicates. When many supervisions are available, the inference process tends to be fully specified (the benchmark assumes that it is always possible to uniquely identify an animal given a complete set of clues, e.g. the values for the intermediate predicated are known), and the gap in the performances of the two collective classifiers becomes smaller.

5.4 Extending Probabilistic Soft Logic

Probabilistic soft logic (PSL) [6, 75] is a general framework for probabilistic reasoning in relational domains (see Sec. 3.3 for more details). Similarly to Markov Logic Networks [120], PSL uses first–order logic rules to instantiate a graphical model having as nodes the values of each grounded predicate, represented as soft-assignments in $[0, 1]$.

PSL uses the Łukasiewicz logic to implement a relaxation technique commonly used to solve MAX SAT problems. In particular, let $C = \{c_1, \dots, c_m\}$ be a set of logic disjunctive clauses, where each formula in the disjunction is a literal, i.e. an atomic formula or its negation. PSL embeds the knowledge into a Markov Random Field (MRF), which builds a distribution over possible interpretations as:

$$P(\mathcal{I}) = \frac{1}{Z} \exp\left(-\sum_{j=1}^{m} \lambda_j \; \Phi_j(\mathcal{I}) \right)$$

where $\lambda_j \geq 0$ is the weight of the clause c_j, Z is the partition function and the potential Φ_j expresses the distance from the satisfaction of the formula c_j. Each weight λ_j can be used to express how strongly the j–th clause is enforced to hold true. In fact, a higher weight penalizes stronger an assignment that does not satisfy the corresponding clause.

PSL assumes the assignment of a template to each clause c_j, that is reused for each single grounding of the clause in the interpretation. Assuming that a clause is universally quantified, the MRF has one clique for each grounding of such formula and the potential Φ_j can be expressed as the sum of the potential ϕ_j on all the possible groundings. In particular, $\Phi_j(\mathcal{I}_j) = \sum_{g \in \mathcal{I}_j} \phi_j(g)$, where $\mathcal{I}_j$ is the set of groundings of the j–th formula with respect to the interpretation $\mathcal{I}$. Let I_j^+ and I_j^- be the indexes of the positive and negative literals respectively in a grounding of c_j. PSL employs the Łukasiewicz logic to express ϕ_j and since disjunctive clauses are supposed, ϕ_j results into the following convex functional:

$$\phi_j(g) = \max\{0, 1 - \sum_{k \in I_j^+} g_k - \sum_{k \in I_j^-} (1 - g_k)\}. \tag{5.15}$$

The PSL framework defines an efficient method to perform inference and to determine the most likely interpretation given the available evidence. This is equivalent to minimize the summation in the exponential, which corresponds to a linear (convex) optimization problem under the restrictions defined above. However, using the concave Łukasiewicz fragment proposed in this paper, inference remains tractable also when lifting the restriction to disjunctive formulas. In fact, as we have already mentioned, any boolean formula can be rewritten in (CNF) and then embedded into the concave fragment exploiting (4.5). However, the negation of such formula, that expresses its distance from the satisfaction to be minimized, can be written as the convex functional:

$$\max\{0, 1 - l_1(g), \ldots, 1 - l_n(g)\}$$

where for $i = 1, \ldots, n$, $l_i(g)$ corresponds to the grounding g of some logic disjunctive clause l_i. Since the expression above can be thought of as a potential that extends (5.15) still preserving convexity, we can extend the set of formulas represented in PSL to the whole concave fragment $(\wedge, \oplus)^*$.

Chapter 6

LYRICS

The theoretical results that have been previously provided, all concern the integration of prior knowledge expressed by first–order logic formulas with machine learning techniques. In this chapter, we present LYRICS (Learning Yourself Reasoning and Inference with ConstraintS), a generic interface layer for AI, which is implemented in TersorFlow (TF) [1]. LYRICS provides a declarative language that allows us to exploit the full expressiveness of first–order logic to define the background knowledge. The predicates and functions of the FOL knowledge can be bound to any TF computational graph, and the formulas are converted into a set of real-valued constraints, which participate to the overall optimization problem. This allows us to learn the weights of the learners, under the constraints imposed by the prior knowledge. The framework is extremely general as it imposes no restrictions in terms of which models or knowledge can be integrated. In the following, we show the potentiality of LYRICS by presenting some use cases of the LYRICS language, including generative models, logic reasoning, model checking and supervised learning.

LYRICS has its root in frameworks like Semantic–Based Regularization (SBR) [33, 34] built on top of Kernel Machines and Logic Tensor Networks (LTNs) [137] that can be applied to neural networks. These frameworks transform the FOL clauses into a set of constraints that are jointly optimized during learning. However, LYRICS generalizes both approaches by allowing us to enforce the prior knowledge transparently at training and test time and dropping all constraints regarding the form of the prior knowledge. Another line of research [32, 126] attempts at using logical background knowledge to improve the embeddings for *relation extraction*. However, these works are also based on ad-hoc solutions that lack a common declarative mechanism that can be easily reused. They are all limited to a subset of FOL and they allow the injection of the knowledge at training time, with no guarantees that the output on the test set satisfies the knowledge. A recent framework to integrate probabilistic logical reasoning with the deep-learning infrastructure of TF is given by TensorLog [19]. However TensorLog is limited to reasoning and does not allow us to optimize the learners while performing inference. Moreover, TensorFlow Distributions [37] and Edward [144] are also related frameworks for integrating probability theory and deep learning, but these frameworks focus on probability theory and not on the representation of logic and reasoning.

The lack of a declarative frontend makes SBR and LTN hard to extend beyond the classical classification tasks, where they have been applied in the past. On the other hand, LYRICS defines a declarative language, which drops the barrier to build models exploiting the available domain knowledge in any machine learning context. In particular, any many-sorted first–order logical theory can be expressed in the framework, allowing us to declare domains of different types, with constants, predicates and functions. LYRICS provides a very tight integration of learning and logic, as any computational graph can be bound to a FOL predicate. This allows us to constrain the learner both during training and inference. Since the framework is agnostic to the learners that are bound to the predicates, the framework can be used in a wide range of applications including classification, generative or adversarial ML, sequence to sequence learning, collective classification, and so on.

6.1 The Declarative Language

LYRICS defines a TensorFlow (TF)[1] environment in which learning and reasoning are integrated. The definition of the knowledge in the presented environment starts by defining a certain number of domains in the considered world. A *domain* determines a collection of individuals of the world that share the same representation space and, thus, can be analyzed and manipulated in a homogeneous way. For example, a domain can collect the set of considered 30×30 pixel images, or the sentences of a book as bag-of-words. The domains are then filled with their "members", on which the learning and reasoning will

[1]`https://www.tensorflow.org/`

be carried.

For example, a domain called *Images* can be defined in LYRICS as:

```
Domain(label="Images", data=data_images)
```

where *data_images* is the placeholder of the input data and *Images* represents the name we may refer to the domain in constraints[2]. The elements of a domain are a sort of "anonymous" individuals that are collectively processed. On the other hand, an *individual* of a domain can also be separately specified, like a FOL constant. In particular, a certain individual can be added to a domain using the following construct:

```
Individual(label="dog12", domains=("Images"), value=img0_data)
```

As we already said, LYRICS allows us to exploit the full expressiveness of first–order logic (see Sec. 2.1.1) to declare the background knowledge. Hence in particular, both FOL functions and predicates are definable. A *function* can be defined to map elements from the input domains into an element of an output domain. A unary function takes as input an element from a domain and transforms it into an element of either the same or of another domain. An n–ary function takes as input n elements (of possibly different domains), mapping them into an element of its output domain. For example, it is possible to define arithmetic functions to operate over numbers, or encoding functions to transform elements of a domain into a latent space. As we will see in Sec 6.4, FOL functions will play a special role in facing generation tasks. In LYRICS, any function is implemented as a TF computational graph, taking as input a fixed number of input tensors representing elements of the input domains and returning an output tensor. The following example shows how to define a function on the domain *Images*:

```
Function(label="encoder", domains=("Images"), function=CNNEncoder)
```

where *CNNEncoder* is the placeholder for the TF implementation of the FOL function named *encoder*. Please note that we decide to not explicitly define the codomain of functions, indeed this is implicitly determined by the domain of predicate functions taking this values as input.

A *predicate* can be defined as a function mapping elements of the input domains to truth values, as for instance *isCat(x)* or *areClose(x,y)*. For example, a predicate *bird* implemented by a neural network NN and taking as input the patterns in the domain *Images* can be defined as:

[2]The attributes *label,data,...* are optional in the declaration and can be omitted.

```
Predicate(label="bird", domains=("Images"), function=NN)
```

Finally, it is possible to declare the knowledge about the world by means of a set of *constraints*. Each constraint is a generic FOL formula built upon the previously defined constructs: individuals, functions and predicates. For instance, if we are given the domain *Images*, composed of animal images, and two predicates *bird* and *flies* defined on it, the user can express the knowledge that all birds fly by means of the constraint:

```
Constraint("forall x: bird(x) -> flies(x)")
```

6.2 From Logic to Learning

The main idea behind the LYRICS framework is that a certain learning problem can be formulated simply declaring a set of (FOL) constraints. However, so far, we only specified the syntax of the language. In this section, we show how this high level description of a problem can be translated into an effective learning model that can be optimized to learn functions and predicates satisfying the given constraints. TensorFlow, which the framework is built on top, performs computations by building a computational graph, where nodes of the graph are operations manipulating all the tensors represented by their incoming edges. TF performs automatic differentiation of a generic computational graph by the exploitation of the chain rule of calculus. Since the framework implements all its components within TensorFlow, any TensorFlow model can be integrated in LYRICS. The proposed framework compiles a high level description of the knowledge into a computational graph by translating each piece of logic knowledge as constraints. The resulting computational graph is optimized exploiting the standard TF optimization mechanism. In the following, we describe how this translation is performed with respect to the general architecture of LYRICS depicted in Figure 6.1.

Domains and Individuals. Domains and individuals allow users to provide data to the framework as tensors and represent the leaves of the computational graph. While domains are represented by constant tensors, individuals can be represented by both *constant* and *variable* tensors. In this way, the user is allowed to provide the knowledge of the existence of a certain

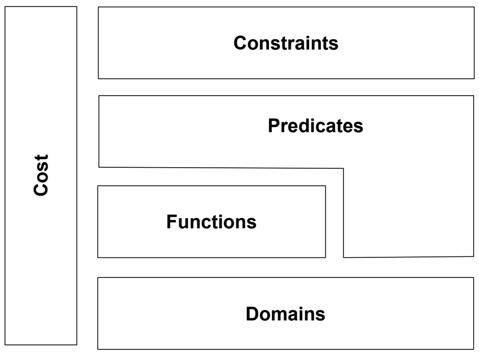

Figure 6.1: *An high level description of the architecture of LYRICS*

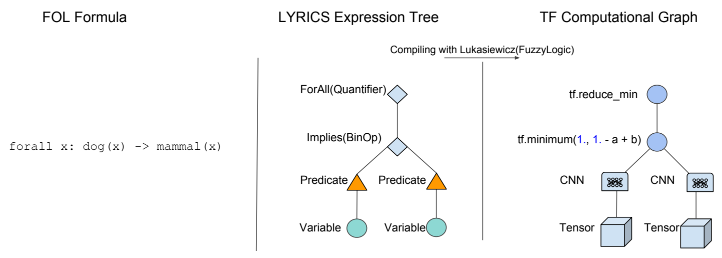

Figure 6.2: *The translation of the FOL formula $\forall x$ dog$(x) \rightarrow$ mammal(x) into a TF computational graph.*

individual, even if its feature representation is unknown, and its representation will be optimized to be coherent with the other provided pieces of knowledge.

Functions and Predicates. Functions allow the mapping among different tensors of (possibly) different domains, while predicates allow us to express the truth degree of some property for those tensors. Both functions and predicates can be implemented using any TF computational graph. If the graph does not contain any variable tensor (i.e. it is not parametric), then we say it to be *given*; otherwise all the variables will be automatically learned to maximize the constraint satisfaction and we say the function/predicate to be *learnable*. Given functions are usually arithmetic functions (e.g. $+$), similarity or comparison functions (e.g. $=, \leq$). Learnable functions can be (deep) neural networks, kernel machines, radial basis functions, or any learning model whose parameters have to be learned. This distinction between given and learnable functions is useful to explain problems but the framework makes no distinction between these two classes.

Constraints. The integration of learning and logical reasoning is achieved by compiling the logical rules into continuous real-valued constraints. The logical rules correlate all the defined elements and enforce some desired behaviors on them. Indeed, the logical formulas combine some facts expressed by the predicates by means of logical connectives with respect to variables and functions applied to them. For this reason, constraints are the main elements of this framework.

Variables, functions, predicates, logical connectives and quantifiers can all be seen as nodes of an *expression tree* [36]. The evaluation of a constraint corresponds to a post-fix visit of the expression tree, where the visit action builds the correspondent portion of computational graph. In particular:

- visiting a *variable* substitutes the variable with a tensor in the domain it belongs to;
- visiting a *function* or *predicate* provides input tensors to the TF models implementing those functions;
- visiting a *connective* combines predicates by means of the real-valued operation associated to the connective;
- visiting a *quantifier* aggregates the outputs of the expressions obtained for the single variable groundings.

Connectives are implemented using fuzzy generalizations of FOL that provide a continuous generalization of their boolean counterparts. However, there are several possible choices to establish their semantics and some possible choices are reported in Table 2.2. The constraints are aggregated over a set of data by means of FOL quantifiers. In particular, the universal and existential quantifiers can be seen as a logic AND and OR applied over each grounding of the data, respectively. Therefore, different quantifiers can be obtained depending on the selection of the underlying t-norm and t-conorm respectively. LYRICS is provided with some built-in logical frameworks, however any user can customize the way connectives and quantifiers are mapped into fuzzy operators. For example, the following code sets the connectives and quantifiers occurring in the constraints to be converted according to Product logic:

```
current_world.logic = LogicFactory.create("product")
```

Finally, any formula is converted by associating to any logical expression its fuzzy representation, as reported for instance in equation (4.1), where the quantifiers are mapped into the *min* and *max* operations. In Figure 6.2 an example of the translation of a logic formula into its expression tree and successively into a TensorFlow computational graph is shown.

For instance consider the rule $\forall x \exists y\; Child(x) \Rightarrow Parent(x,y) \wedge Mother(y)$ where $Child$, $Mother$ are unary predicates determining whether a pattern is a child and a mother, respectively and $Parent$ is a binary predicate indicating whether a pair of data points are linked via a parental link. This rule states that every child has a parent who is her/his mother. Using the Product t-norm both for the conjunction and the universal quantifier, the material implication of Product logic (see Table 2.2) and a *max* operation for the existential quantifier, the rule is translated as follows:

$$\prod_{x \in \mathcal{X}} \max_{y \in \mathcal{X}} \left\{ 1 - f_C(x) + f_C(x) \cdot f_P(x,y) \cdot f_M(y) \right\}$$

where f_C, f_M, f_P are functions (to be learned) approximating the predicates $Child$, $Mother$ and $Parent$, respectively and $\mathcal{X}$ is the set of patterns representing a group of people.

Supervisions Any available supervision for the functions or predicates can be integrated into the learning problem. LYRICS provides a placeholder where this fitting is expressed, called *PointwiseConstraint*. This construct refers to a computational graph where a loss is applied for each supervision (e.g. the loss defaults to the cross–entropy loss for any neural network classifier) and it can be overridden to achieve a different behavior:

```
PointwiseConstraint(model, labels, inputs)
```

where *model* is the function for which to enforce supervisions *labels* on data *inputs*. However, as we already observed in Sec. 5.2.2, the pointwise contraints can also be written directly in logical form.

Cost Function Let us assume to be given a knowledge base consisting of a set of FOL formulas $KB = \{\varphi_1, \dots, \varphi_H\}$, where some of the elements (individuals, functions or predicates) are unknown. The learning process aims at finding a good approximation of each unknown element, so that the estimated values will satisfy the FOL formulas for the input samples. A computational graph for each constraint is built. Let $\boldsymbol{f}$ and $\boldsymbol{p}$ be the vectors of learnable functions and predicates, respectively. Let $0 \leq \Phi_h(\mathcal{X}_h, \boldsymbol{f}, \boldsymbol{p}) \leq 1$ indicate the degree of satisfaction of the h-th constraint evaluated on all its inputs $\mathcal{X}_h$. This framework is very general and it accommodates learning from examples as well as the integration with FOL knowledge. In general terms, the learning scheme we propose can be formulated as the minimization of the following term in order to satisfy the constraints:

$$L_c(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p}) = \sum_{h=1}^{H} \lambda_h^c \mathcal{L}\Big(\Phi_h\big(\mathcal{X}_h, \boldsymbol{f}, \boldsymbol{p}\big)\Big) \;, \tag{6.1}$$

where $\mathcal{X}$ denotes the overall set of samples where the functions and predicates are evaluated, λ_h^c denotes the weight for the h–th logical constraint and the function $\mathcal{L}$ represents any monotonically decreasing transformation of the constraints conveniently chosen according to the problem under investigation. Common choices adopted in LYRICS for $\mathcal{L}$ are given by:

$$(\mathbf{a}) \qquad \mathcal{L}\Big(\Phi_h\Big) = 1 - \Phi_h, \tag{6.2}$$

$$(\mathbf{b}) \qquad \mathcal{L}\Big(\Phi_h\Big) = -\log\Big(\Phi_h\Big) \;. \tag{6.3}$$

The conversion of formulas into real-valued constraints is carried out automatically in the framework we propose. Indeed, LYRICS takes as input the expressions defined using a declarative language and builds the constraints once we decide the conversion functions to be exploited. For instance, when the mapping defined in Equation 6.2-**(b)** is applied to an universally quantified formula, i.e. $\forall x\, \varphi(x)$, it yields the following constraint:

$$\begin{aligned} \mathcal{L}\left(\prod_{x\in\mathcal{X}} \Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})\right) &= -\log\left(\prod_{x\in\mathcal{X}} \Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})\right) \\ &= \sum_{x\in\mathcal{X}} -\log\big(\Phi(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p})\big) \;, \end{aligned} \tag{6.4}$$

that corresponds to a generalization to generic fuzzy logic expressions of the cross–entropy loss, which is commonly used to force the fitting of the supervised data for deep learners.

However, LYRICS allows us to integrate classical supervised learning and learning from constraints modeling the prior knowledge. The overall cost function may be composed of different terms, both on functions and predicates, forcing the fitting of the supervised examples, avoiding complexity in the learned functions and expressing the logical constraints satisfaction:

$$\lambda_s^f L_s(\mathcal{X}_s^f, \boldsymbol{f}) + \lambda_s^p L_s(\mathcal{X}_s^p, \boldsymbol{p}) + \lambda_r^f L_r(\boldsymbol{f}) + \lambda_r^p L_r(\boldsymbol{p}) + \lambda_c L_c(\mathcal{X}, \boldsymbol{f}, \boldsymbol{p}) \ ,$$

where $\mathcal{X}_s^f, \mathcal{X}_s^p \in \mathcal{X}$ denote the supervised data for functions and predicates respectively, L_s is a loss function to enforce the supervisions, L_r expresses a regularization term and $\lambda_s^f, \lambda_s^p, \lambda_r^f, \lambda_r^p, \lambda_c$ denote some coefficients weighting (differently) each loss contribution.

6.3 Example Driven Presentation of LYRICS

Here we present a list of examples illustrating the range of learning tasks involving both symbolic and sub-symbolic information that can be expressed in the proposed framework. In particular, it is shown how it is possible to force label coherence in semi–supervised or transductive learning tasks, how to implement collective classification over the test set, rule induction from the learned predicates as in classical inductive logic programming (ILP), pure logical reasoning and how to address generative tasks or pattern completion in the case of missing features. The examples are presented using the LYRICS syntax directly to show that the final implementation of a problem fairly retraces its abstract definition[3].

Semi–Supervised Learning

In this task we assume to have available a set of 420 points distributed along an outer and an inner circle. The inner and outer points belong and do not belong to some given class A, respectively. A random selection of 20 points is supervised (either positively or negatively), as shown in Figure 6.3. The remaining points are split into 200 unsupervised training points, shown in

[3]The software of the framework and the experiments are made available at `https://github.com/GiuseppeMarra/lyrics`.

Figure 6.3 and 200 points left as test set. In what follows, we assume that a neural network has been created in TF to approximate the predicate A.

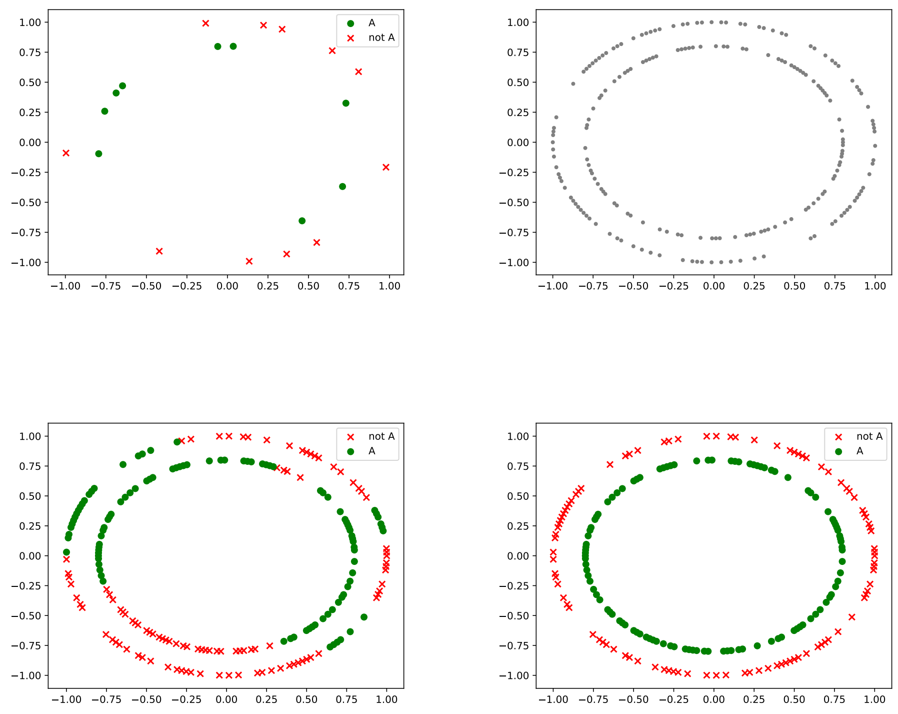

Figure 6.3: *Semi–supervised Learning: (a) data that is provided with positive and negative supervisions for class A; (b) the unsupervised data provided to the learner; (c) class assignments using only the supervised examples; (d) class assignments using learning from examples and constraints.*

The network can be trained requiring the fitting with the supervised data. Hence, given the vector of data X, a neural network *NN_A* and the vector of supervised data *X_s* with the vector of associated labels *y_s*, the supervised training of the network can be expressed by the following:

```
# Definition of the data points domain.
Domain(label="Points", data=X)
# Approximating the predicate A via a NN.
Predicate(label="A", domains=("Points"), function=NN_A)
# Fit the supervisions
```

```
PointwiseConstraint(NN_A, y_s, X_s)
```

Let's now assume that we want to express manifold regularization (see equation (4.6)) with respect to a certain closeness relation for the learned function, i.e. points that are close should be similarly classified. This can be expressed as:

```
# Predicate stating whether two patterns are close.
Predicate("Close", ("Points","Points"), f_close)
# Manifold regularization constraint.
Constraint("forall p:forall q: Close(p,q)->(A(p)<->A(q))")
```

where *f_close* is a given function determining if two patterns are close. The training is then re-executed starting from the same initial conditions as in the supervised-only case. Figure 6.3 shows the class assignments of the patterns in the test set, when using only learning from supervised examples. Finally, Figure 6.3 presents the assignments when learning from examples and constraints.

Collective Classification

Collective classification (see Sec. 5.3) performs the class assignments exploiting any known correlation among the test patterns. Here we show how to exploit these correlations in LYRICS. We assume that the patterns are represented as $\mathbb{R}^2$ datapoints. The classification task is a multi-label problem where the patterns belong to three classes A, B, C. In particular, the class assignments are defined by the following membership regions:

$$\mathbf{A} = [-2,1] \times [-2,2], \mathbf{B} = [-1,2] \times [-2,2], \mathbf{C} = [-1,1] \times [-2,2] \ .$$

These regions correspond to three overlapping rectangles as shown in Figure 6.4-(**a**). The examples are partially labeled and drawn from a uniform distribution on both the positive and negative regions for all the classes. In a first stage, the classifiers for the three classes are trained in a supervised fashion using a two-layer neural network taking four positive and four negative examples for each class. This is implemented via the following declaration:

```
Domain(label="Points", data=X)
Predicate(label="A",domains=("Points"),NN_A)
Predicate(label="B",domains=("Points"),NN_B)
```

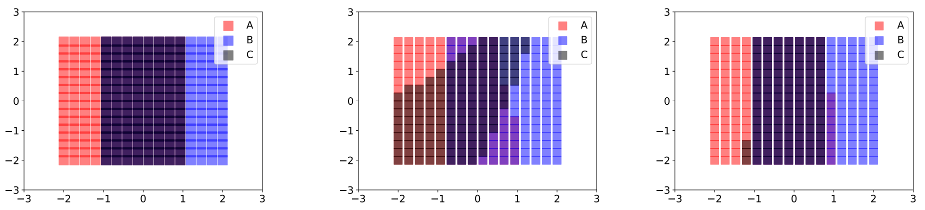

Figure 6.4: *Collective classification: (a) class assignments; (b) the predictions after the supervised step; (c) the predictions with collective classification and rule satisfaction.*

```
Predicate(label="C",domains=("Points"),NN_C)
PointwiseConstraint(NN_A, y_A, X_A)
PointwiseConstraint(NN_B, y_B, X_B)
PointwiseConstraint(NN_C, y_C, X_C)
```

The test set is composed by 256 random points and the assignments performed by the classifiers after the training are reported in Figure 6.4-(**b**). In a second stage, some prior knowledge about the task is exploited. In particular, any pattern must belongs to (at least) one of the classes A or B. Furthermore, it is known that class C is defined as the intersection of A and B. The classifiers trained in the first stage provide some initial predictions for each pattern, while the collective classification step is performed by seeking the class assignments that are as close as possible to the initial classifier predictions but also respect the logical constraints on the test set:

```
Constraint("forall x: A(x) or B(x)")
Constraint("forall x:(A(x) and B(x)) <-> C(x)")
# Minimize the distance from prior values
PointwiseConstraint(NN_A, priorsA, X_test)
PointwiseConstraint(NN_B, priorsB, X_test)
PointwiseConstraint(NN_C, priorsC, X_test)
```

where *X_test* denotes the set of test datapoints and *priorsA*, *priorsB*, *priorsC* denote the outputs of the classifiers acting as priors for the final assignments. As we can see from Figure 6.4-(**c**), the collective step fixes some wrong predictions.

Rule induction and model checking

It is also possible to mark a set of constraints as test only, in order to perform model checking. Model checking can be used as a fundamental step to perform rule induction using the ILP techniques [109]. In this example, we show how the framework can be used to infer rules and perform model checking. A brute force approach is applied here, but the inductive logic programming community has designed more scalable algorithms that could be exploited [25]. Let us consider a simple multi-label classification task where the patterns belong to two classes A and B, and B is contained in A. This case models a simple hierarchical classification task. In particular, the classes are defined by the following membership regions: $\mathbf{A} = [-2, 2]\times[-2, 2]$, $\mathbf{B} = [-1, 1]\times[-1, 1]$. A set of points X is drawn from a uniform distribution in the $[-3, 3]\times[-3, 3]$ region. Two neural network classifiers are trained to classify the points:

```
Domain(label="Points", data=X)
Predicate(label="A", domains=("Points"), NN_A)
Predicate(label="B", domains=("Points"), NN_B)
PointwiseConstraint(NN_A, y_A, X)
PointwiseConstraint(NN_B, y_B, X)
```

It could be interesting to discover which logical relations are learned by the classifiers. In order to achieve this goal, we build all possible formulas in Disjunctive Normal Form (DNF) that are universally quantified with a single variable. One constraint is built and evaluated for each formula, and LYRICS measures the degree of satisfaction of the rule over the data. In the considered task, an example of a constraint getting a very high degree of satisfaction is:

```
Constraint("forall x: (not A(x) and not B(x)) or (A(x) and not B(x))
    or (A(x) and B(x))")
```

As one could expect, the only fully-satisfied constraint is given by the following FOL formula $\forall x \neg B(x) \vee A(x)$, or for simple $\forall x B(x) \rightarrow A(x)$, that states the inclusion of B in A.

Logic Reasoning

The presented framework can be used as a tool for pure logical reasoning. This case is illustrated by the following example, where few individuals are separately added to the domain *People* without any underlying data representation (no features) by the statement:

```
Domain(label="People")
Individual(label="Marco", "People")
Individual(label="Giuseppe", "People")
Individual(label="Michele", "People")
Individual(label="Francesco", "People")
Individual(label="Franco", "People")
Individual(label="Andrea", "People")
```

The individuals are assumed to be related via parental relations defined by the following predicates, where the given binary predicate *eq* holds true if and only if the two input individuals are the same person:

```
Predicate("father", ("People","People"))
Predicate("grandFather", ("People","People"))
Predicate("eq", ("People","People"), eq)
```

In addition, some known relations are known among the individuals:

```
Constraint("father(Marco, Giuseppe)")
Constraint("father(Giuseppe, Michele)")
Constraint("father(Giuseppe, Francesco)")
Constraint("father(Franco, Andrea)")
```

The prior knowledge provided for this task expresses some well-known semantics about parental constraints. For example, it is possible to express that nobody can be father or grandfather of himself as:

```
Constraint("forall x: not father(x,x)")
Constraint("forall x: not grandFather(x,x)")
```

Other two rules state that fathership is an asymmetric relation, so that if you are father or grandfather of someone, he can not be your father or grandfather. Furthermore, someone cannot be father and grandfather of someone at the same time, these are expressed as:

```
Constraint("forall x: forall y: father(x,y) -> not father(y,x)")
Constraint("forall x: forall y: grandFather(x,y) -> not grandFather(
    y,x)")
Constraint("forall x: forall y: father(x,y) -> not grandFather(x,y)"
    )
Constraint("forall x: forall y: grandFather(x,y)->not father(x,y)")
```

Other rules express that the father of the father is a grandfather, and that one person has at most one father in the considered world:

```
Constraint("forall x: forall y: forall z: father(x,z) and father(z,y
    ) -> grandFather(x,y)")
Constraint("forall x: forall y: forall z: (father(x,y) and not eq(x,
    z)) -> not father(z,y)")
```

The learning task seeks to infer the unknown relations among the individuals. After starting the learning phase, the predicate values are returned for all the groundings and some facts are correctly concluded. For instance:

$$\begin{gathered} grandFather("Marco","Michele")\ , \\ \neg grandFather("Marco","Giuseppe")\ , \\ grandFather("Marco","Francesco")\ , \\ \vdots \end{gathered}$$

On the other hand, nothing can be concluded regarding who is the grandfather of "Franco" and "Andrea", so leaving these values to be equal to their prior values. Once the training has been performed and the grounded predicates have been computed, model checking can be performed by stating the rules that should be verified. For example:

```
Constraint("forall x: forall y: forall z: grandFather(x,z) and
father(y,z) -> father(x,y)")
```

As expected, the evaluation of the rule returns that it is perfectly verified by the computed assignments.

Missing Features

A set of patterns are drawn from a double moon shaped distribution as shown in Figure 6.5-(**a**). The patterns distributed along the lower moon belong to class A, while patterns along the higher one do not. This task is expressed as:

```
Domain(label="Points", data=X)
Predicate("A", "Points", NN_A)
PointwiseConstraint(NN_A, y_A, X)
```

Let us now assume that there are two new individuals $p0$ and $p1$ for which no feature representation is available, but it is known that $p0$ and $p1$ belong and do not belong to class A, respectively. This can be expressed as:

```
Individual(label="p0", ("Points"))
Individual(label="p1", ("Points"))
Constraint("A(p0)")
Constraint("not A(p1)")
```

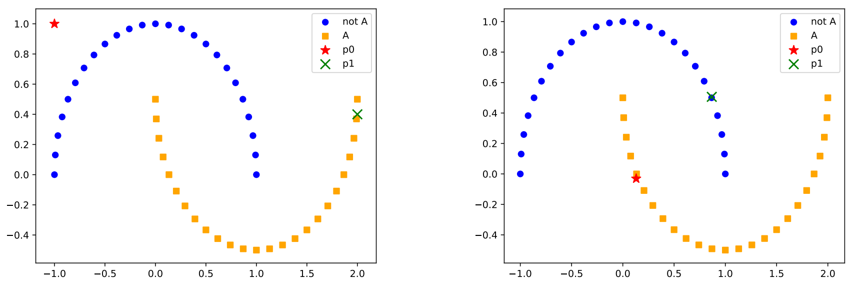

Figure 6.5: *Missing Features: (a) data provided with positive or negative supervision for class A and initial random positions of points p_0 and p_1 (b) learned final positions of p_0 and p_1, after the further constrains are enforced.*

We assume to know in advance that the two individuals are close to a positive and a negative example for class A, respectively. This can be stated as:

```
Predicate("Close", ("Points","Points"), close)
Predicate("eq", ("Points", "Points"), eq)
Constraint("exists q: not eq(q,p0) and A(q) and Close(q,p0)")
Constraint("exists q: not eq(q,p1) and not A(q) and Close(q,p1)")
```

where *close* is a given predicate deciding whether two points are close and *eq* implements a differentiable equality function defined by: $1 - tanh(||x - y||^2)$.

Since the feature representations of $p0$ and $p1$ are not defined and are left as free variables, the framework can learn them in order to respect the constraints defined by the logic rules. Figure 6.5-(**b**) shows the values of individuals after training. They have been correctly placed, where the data distribution of the corresponding classes of the points is high.

6.4 Generative Learning and Visual Translation

So far, we introduced the syntax of LYRICS, a general interface layer for AI, and we provided a list of examples in different learning and reasoning scenarios to enlighten its potential expressiveness. In this section, we propose a general approach to visual generation that combines learning capabilities with logic descriptions of the target to be generated. The process of generation is regarded as a constrained satisfaction problem, where the constraints describe a set of properties that characterize the target. In particular, we propose an example of pattern generation and the modeling of Generative Adversarial Networks (GANs) [56] reporting promising results in image translation of human faces. GANs have achieved impressive results in image generation. By taking inspiration from the Turing test [146], a generator function is asked to fool a discriminator function which, in turn, tries to distinguish real samples from generated ones. This mechanism allows GANs to generate very realistic images when trained properly.

The systematic adoption of deep learning to visual generation has recently produced impressive results that, amongst others, definitely benefit from the massive exploration of convolutional architectures. A special generation task is image-to-image translation, which learns to map each image of an input domain into an image in a (possibly different) output domain. In most real-world domains, there are no pairs of examples showing how to translate an image into a corresponding one in another domain, yielding the so called UNsupervised Image-to-image Translation (UNIT) problem. In an UNIT problem, two independent sets of images belonging to two different domains (e.g. cats-dogs, male-female, summer-winter, etc.) are given and the task is to translate an image from one domain into the corresponding image in the other domain, even though there exist no paired examples showing this mapping. Unfortunately, estimating a joint distribution of the images in the two domains from the distributions in the original single domains is known to have infinite possible solutions. Therefore, one possible strategy consists in mapping pairs of corresponding images to the same latent space using auto-encoders and then learning to reconstruct an image from its representation in latent space. Combining auto-encoders with GANs has been proposed in [87, 127] and outstanding results on image translation have been reported by [90, 91, 155].

In the following, we propose a general approach to visual generation and translation that combines learning capabilities with logic descriptions of the images that are generated. The generation problem is translated into a constrained satisfaction problem, where each constraint forces the generated image to have some predefined feature. A main advantage of this approach is to decouple the logic description level from the generative models. The logic layer is architecture agnostic, allowing us to inject any generator model based on deep learning into the logic layers. In particular, expressing the task using logic knowledge allows us to easily extend the involved classes to additional translation categories as well as yielding an easier to understand learning scheme. The translations are then interleaved and jointly learned using the constraints generated by the framework that allow us to obtain truly realistic images on different translation types. Finally, the experiments show how to formulate an image-to-image task using logic, including adversarial tasks with cycle consistency. The declarative approach allows us to easily interleave and jointly learn an arbitrary number of translation tasks.

6.4.1 Expressing an Adversarial Setting by FOL

Here we show how the discriminative and generative parts of an image-to-image translation system can be formulated by merging logic and learning, yielding a more understandable and easier to extend setup. Let us assume to be given a set of images $\mathcal{I}$. There are two components of a translator framework to be considered. First, a set of *generator* functions $g_j : \mathcal{I} \to \mathcal{I}$, which take as input an image representation and return a corresponding image in the same output domain, depending on the semantics given to the task. Second, a set of *discriminator* functions $d_i : \mathcal{I} \to [0, 1]$ determining whether an input image $x \in \mathcal{I}$ belongs to class i and, thus, they must be intended in a more general way than in traditional GANs. Interestingly, all learnable FOL functions (i.e. functions mapping input elements into an output element) can be interpreted as generator functions and all learnable FOL predicates (i.e. functions mapping input elements into a truth value) can be interpreted as discriminator functions.

The discriminators can be trained by providing some examples in the original domains as:

$$\forall x \, S_i(x) \Rightarrow d_i(x), \quad i = 1, 2, \ldots$$

where $S_i(x)$ is a given function returning true if and only if an image is a positive example for the i–th discriminator. These constraints allow us to transfer the knowledge provided by the supervision (i.e. the $S_i(x)$) inside the discriminators that are learnable functions.

Moreover, $d_i(x)$ functions are differentiable and can be exploited to train the generators functions. To this end, assuming that a generator function has to produce an image with a certain property, we can force the corresponding discriminator function for such a property to positively classify it. Therefore, assuming that the goal of the j–th generator is to generate images of class j, this can be typically expressed by a rule taking the form:

$$\forall x \, d_j(g_j(x)), \quad j = 1, 2, \ldots$$

In perspective, the logical formalism could provide a simple way to describe complex behaviors of generator functions by interleaving multiple positive or negative discriminative atoms. By requiring that a generated image should appear realistic, the GAN framework implements a special case of these constraints, where the required property is the similarity with real images.

Cycle consistency [155] is also commonly employed to impose that by translating an image from a domain to another one and then translating it back to the first one, we should recover the input image. Cycle consistency allows us to restrict the number of possible translations, especially when concatenating different generative functions. This can be formulated in FOL as:

$$\forall x \;\; S_i(x) \Rightarrow g_i(g_j(x)) = x, \quad i = 1, 2, \ldots, \quad j = 1, 2, \ldots$$

Clearly, in complex problems, the chain of functions intervening in these constraints can be longer, but the main idea remains exactly the same.

Sometimes, images belonging to different classes are required to share the same common latent space. Let us indicate $e : \mathcal{I} \rightarrow \mathbb{R}^n$ an encoding function mapping the image into a latent space, that has to be jointly learned during the learning phase. In this special case, the generators must be re-defined as decoder functions taking as input the latent representation of the images, namely: $g_j : \mathbb{R}^n \rightarrow \mathcal{I}$. The auto-encoding constraints can be expressed as:

$$\forall x \; S_i(x) \Rightarrow g_i(e(x)) = x, \quad i = 1, 2, \ldots$$

Up to now, the described constraints are very general and they can be exploited in almost all generative translation tasks. However, as we will see in the next section, the logical formalism (and the LYRICS environment) allows the enforcement of any additional knowledge about the task.

6.4.2 Experimental Results

In this section, we show how to effectively setting up some generation (and eventually adversarial) tasks in LYRICS, exploiting the constraints schema described in the previous section about generative and discriminative functions. In what follows, in order to not overload the notation, instead of expressing the constraints in the LYRICS syntax, we directly make use of FOL.

Next and Previous Digits Generation

As a toy example, we show a task in which we are asked to learn two generative functions, *next* and *previous*, which, given an image of a $0, 1, 2$ digit, return an image of a digit that is respectively, the next and previous with respect to the natural number order. In order to give each image a next and a previous digit in the chosen set, a circular mapping was used such that 0 is the next digit of 2 and 2 is the previous digit of 0.

The functions *next* and *previous* are implemented by feedforward neural networks with 50 neurons and 1 hidden layer. Since the outputs of such functions are still images, the output size of the networks is equal to the input size. A 1-hidden layer radial basis function (RBF) with a 3-sized softmax output layer is used to implement the *zero*, *one* and *two* discriminators bound to the three outputs of the network, respectively. The RBF model, by constructing closed decision boundaries, allows the generated images to resemble the input ones. Finally, let $isZero$, $isOne$ and $isTwo$ be three given functions, defined on the input domain, returning 1 only if an image is a 0, 1 or 2, respectively. They play the role of the functions $S_i(x)$ that have been previously described.

The idea is to solve both a classification and a generation task. The first one aims at identifying which digit an image represents, while the generation task aims at learning generative functions without giving any direct supervision to them, but simply requiring that the generation is consistent with the classification performed by the jointly learned classifiers.

The classification problem can be described by the following constraints to learn the discriminators

$$\begin{gathered}\forall x\, isZero(x) \Rightarrow zero(x)\\ \forall x\, isOne(x) \Rightarrow one(x)\\ \forall x\, isTwo(x) \Rightarrow two(x)\end{gathered}$$

while the following ones express that the generative functions are constrained to return images which are correctly recognized by the discriminators

$$\begin{gathered}\forall x\ zero(x) \Rightarrow one(next(x)) \wedge two(previous(x))\\ \forall x\ one(x) \Rightarrow two(next(x)) \wedge zero(previous(x))\\ \forall x\ two(x) \Rightarrow zero(next(x)) \wedge one(previous(x))\end{gathered}$$

In addition, in order to force the generated images to be more realistic, we can ask they are similar to at least one existing digit in the domain, by enforcing the following constraints:

$$\begin{gathered}\forall x\ \exists y\ (isZero(x) \wedge isOne(y)) \Rightarrow next(x) = y\\ \forall x\ \exists y\ (isZero(x) \wedge isTwo(y)) \Rightarrow previous(x) = y\\ \forall x\ \exists y\ (isOne(x) \wedge isTwo(y)) \Rightarrow next(x) = y\\ \forall x\ \exists y\ (isOne(x) \wedge isZero(y)) \Rightarrow previous(x) = y\\ \forall x\ \exists y\ (isTwo(x) \wedge isZero(y)) \Rightarrow next(x) = y\\ \forall x\ \exists y\ (isTwo(x) \wedge isOne(y)) \Rightarrow previous(x) = y\end{gathered}$$

Finally, the cycle consistency constraints for the digit generators can be expressed by:

$$\begin{gathered}\forall x\, next(previous(x)) = x\\ \forall x\, previous(next(x)) = x\ .\end{gathered}$$

The equality operator can be validated according to the considered task. In these constraints, we take a given binary predicate computing a pixel by pixel similarity between the images and defined as $1 - \tanh(\frac{1}{P}\sum_p |x_p - y_p|)$ where x_p and y_p are the p-th pixels of the images x, y.

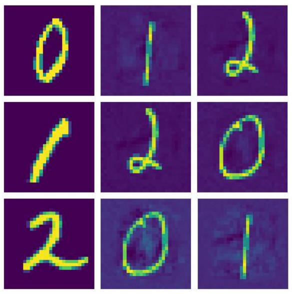

Figure 6.6: *An example of the trained generative functions. The first column pictures represents the input images. The second and third column pictures show the outputs of the functions* next *and* previous*, respectively, computed on the input image.*

We test this idea by taking a set of around 15000 images of handwritten characters, obtained extracting only the 0, 1 and 2 digits from the MNIST dataset. The mentioned constraints have been expressed in LYRICS and the model computational graphs have been bound to the predicates. Figure 6.6 shows an example of image translation using this schema, where the image on the left is an original MNIST image and the two right images are the output of the *next* and *previous* generators. The simplicity of the task, the particular discriminators models and the availability of a rich knowledge about the images domain make it possible for the generative functions to produce realistic images without the need of an adversarial approach. This will not be the case of the experiments presented in the next section where an adversarial approach is, instead, indispensable and its use appears to be naturally incorporated into our logical framework.

Before proceeding, we want to dwell on the possibilities of this approach after an example has been provided. The declarative nature of the logical formalism and its subsequent translation into real-valued constraints, exploited as loss functions of an optimization problem, enables the construction of very complex generative problems by means of only a high-level semantic description. By exploiting models inherited from the literature, a final user is allowed to face the most different problems with the minimum implementation effort.

Experiments on Image Translation

In the following, we show a real image-to-image translation task applying the general setup described in the previous section, including auto-encoders, GANs and cycle consistency. The declarative nature of the formulation makes it very easy to add an arbitrary number of translation problems and it allows us to easily learn them jointly.

UNIT translation tasks assume that there are no pairs of examples showing how to translate an image into a corresponding one in another domain. Combining auto-encoders with GANs is the state-of-the-art solution for tackling UNIT generation problems [90,91,155]. In this section, we show how this state-of-the-art adversarial setting can be naturally described and extended by the proposed logical and learning framework. Furthermore, we show how the logical formulation allows a straightforward extension of this application to a greater number of domains. The CelebFaces Attributes dataset [92] was used to evaluate the proposed approach, where celebrities face images are labeled with various attributes gender, hair color, smiling, eyeglasses, etc. Images are defined as 3D pixel tensors with values belonging to the $[0, 1]$ interval. The first two dimensions represent width and height coordinates while the last dimension indexes among the RGB channels.

Gender Translation We used the *Male* attribute to divide the entire dataset into two input categories, namely male and female images. In the following $S_M(x)$ and $S_F(x)$ (such that $\forall x \; S_F(x) \Leftrightarrow \neg S_M(x)$) are two given predicates holding true if and only if an image x is or is not tagged with the *male* tag. Let e be an encoding function mapping images into the latent domain $\mathcal{Z} = \mathbb{R}^n$. The encoders are implemented as multilayer convolutional neural networks with resblocks [65], leaky-ReLU activation functions and instance normalization at each layer (see [90] for a detailed description of the architecture). The generative functions g_M and g_F map vectors of the domain $\mathcal{Z}$ into images. These functions are implemented as multilayer transposed convolutional neural networks (also called "deconvolutions") with resblocks, leaky-ReLU activation functions and instance normalization at each layer. To implement the shared latent space assumption, g_M and g_F share the parameters of the first layer. The functions d_M and d_F are trained to discriminate whether an image is real or it has been generated by the g_M and g_F generator functions. For example, if x and y are two images such that $S_M(x), S_F(y)$ hold true, then $d_M(x)$ should return 1 while $d_M(g_M(e(y)))$ should return 0. The architectures of the models implementing e, d_M, d_F, g_M, g_F are replicated from some state-of-the-art models [90,91,155]. All these papers show that the use of convolutional models in conjunction with resblocks and instance normalization allows us to obtain truly realistic and high definition images.

The problem can be described as follows. First, we look at the constraints that the encoding and generative functions need to satisfy. We ask the encoder and the generator of the same domain to be circular, that is to map the input into itself, as in the autoencoding scheme proposed by Liu et al. [90]:

$$\forall x\ S_M(x) \Rightarrow g_M(e(x)) = x \tag{6.5}$$

$$\forall x\ S_F(x) \Rightarrow g_F(e(x)) = x \tag{6.6}$$

where the equality operator comparing two images is bound to a continuous and differentiable function computing a pixel by pixel similarity between the images, defined as $1-\tanh(\frac{1}{P}\sum_p |x_p - y_p|)$ where x_p and y_p are the p-th pixels of the x and y images and P is the total number of pixels.

Cycle consistency is also imposed as described in the previous section as:

$$\forall x\ S_M(x) \Rightarrow g_M(e(g_F(e(x)))) = x \tag{6.7}$$

$$\forall x\ S_F(x) \Rightarrow g_F(e(g_M(e(x)))) = x \tag{6.8}$$

where the same equality operator is used to compare the images.

Finally, the generated images must fool the discriminators so that they will be detected as real ones as:

$$\forall x\ S_M(x) \Rightarrow d_F(g_F(e(x))) \tag{6.9}$$

$$\forall x\ S_F(x) \Rightarrow d_M(g_M(e(x))) \tag{6.10}$$

On the other hand, the discriminators must correctly discriminate real images from generated ones by the satisfaction of the following constraints:

$$\forall x\ S_M(x) \Rightarrow d_M(x) \wedge \neg d_F(g_F(e(x))) \tag{6.11}$$

$$\forall x\ S_F(x) \Rightarrow d_F(x) \wedge \neg d_M(g_M(e(x))) \tag{6.12}$$

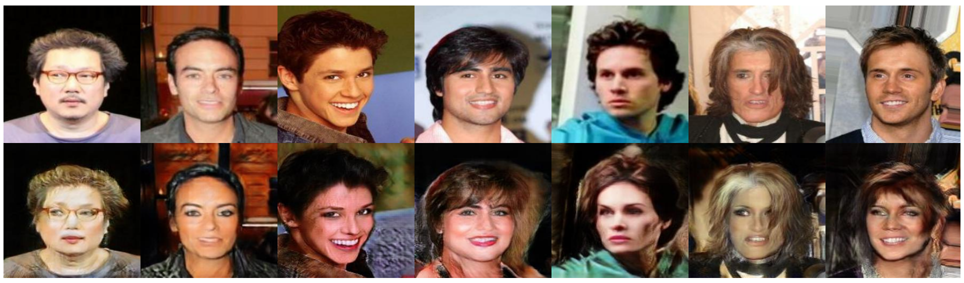

Figure 6.7: ***Face Gender Translation: male to female.*** *The top row shows input male images whereas the bottom row shows the corresponding generated female images.*

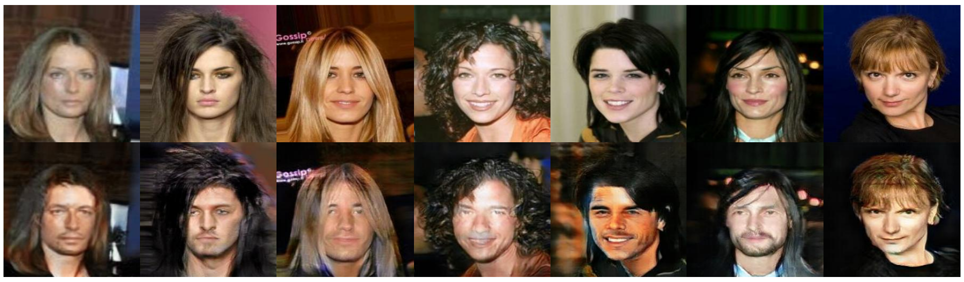

Figure 6.8: ***Face Gender Translation: female to male.*** *The top row shows input female images whereas the bottom row shows the corresponding generated male images.*

Using logical constraints allows us to give a clean and easy formulation of the adversarial setting. Indeed, the adversarial constraints can be interpreted as all others constraints which exploit classification functions. These constraints force the generative function to produce samples that are categorized in the desired class by the discriminator. Moreover, the decoupling between the models used to implement the functions and the description of the problem makes really straightforward to extend or transfer this setting.

The task is implemented in LYRICS exploiting the Product logic to convert the connectives and the quantifiers aggregating all the groundings in the FOL formulas described so far. As we saw (see equation (6.4)), the selection of this t-norm is particularly suited for this task because it defines a cross-entropy loss on the output of the discriminators, which is the loss that was also used to train these models in their original setup. The functions e, g_M and g_F

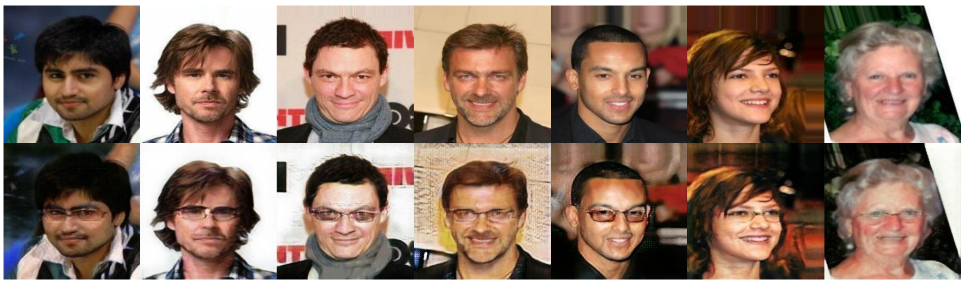

Figure 6.9: ***Face Gender Translation: male/female to eyeglasses.*** *The top row shows input male/female images whereas the bottom row shows the correspondent generated faces with eyeglasses.*

are trained to satisfy the constraints defined in (6.5)–(6.10), while the functions d_M and d_F are trained to satisfy the constraints (6.11), (6.12). Weight learning for the models was performed using the Adam optimizer with a fixed learning rate equal to 0.0001. Figures 6.7 and 6.8 show some male-to-female and female-to-male translations, respectively.

Adding Eyeglasses Given this setting, we can integrate a third domain in the overall problem adding the corresponding constraints for this class. Let $S_E(x)$ be a given predicate holding true if and only if an image x is tagged with the *eyeglasses* tag in the dataset. Let $g_E(x)$ be the corresponding generator and $d_E(x)$ the corresponding discriminator for this property. The same network architectures of the previous description are employed to implement d_E and g_E. The addition of this third class requires to add the following constraints for the generators, to be integrated with the male and female classes,

$$
\begin{aligned}
&\forall x\ S_M(x) \Rightarrow d_E(g_E(e(x)))\\
&\forall x\ S_F(x) \Rightarrow d_E(g_E(e(x)))\\
&\forall x\ S_E(x) \Rightarrow g_E(e(x)) = x\\
&\forall x\ S_M(x) \wedge S_E(x) \Rightarrow d_E(g_F(e(x)))\\
&\forall x\ S_F(x) \wedge S_E(x) \Rightarrow d_E(g_M(e(x)))\\
&\forall x\ S_M(x) \wedge S_E(x) \Rightarrow g_E(e(g_F(e(x))) = g_F(e(x))\\
&\forall x\ S_F(x) \wedge S_E(x) \Rightarrow g_E(e(g_M(e(x))) = g_M(e(x))\\
&\forall x\ S_M(x) \wedge \neg S_E(x) \Rightarrow g_M(e(g_E(e(x))) = g_E(e(x))\\
&\forall x\ S_F(x) \wedge \neg S_E(x) \Rightarrow g_F(e(g_E(e(x))) = g_E(e(x))
\end{aligned}
$$

and to add the following for the discriminator:

$$
\begin{aligned}
&\forall x\ S_E(x) \Rightarrow d_E(x)\\
&\forall x\ S_M(x) \wedge \neg S_E(x) \Rightarrow \neg d_E(g_E(e(x)))\\
&\forall x\ S_F(x) \wedge \neg S_E(x) \Rightarrow \neg d_E(g_E(e(x)))
\end{aligned}
$$

We note that in this case, the class eyeglasses is not mutually exclusive neither with male nor female class. This is the reason why we have to consider some constraints with a conjunction on premises. In addition, we have to distinguish how the male and female generators behave in presence of the attribute eyeglasses. In particular we enforce that translating a gender attribute does not affect the presence of eyeglasses. Figure 6.9 shows some examples of the original face images, and the corresponding generated images of the faces with added eyeglasses.

As we already said, the proposed approach is very general and can be exploited to manage possibly several attributes in a visual generation task combining a high-level logical description with deep neural networks. The most distinguishing property is the flexibility of describing new generation problems by simple logic descriptions, which leads to attack very different problems. Instead of looking for specific hand-crafted cost functions, the proposed approach offers a general scheme for their construction that arises from the t-norm theory. Moreover, the interleaving of different image translations tasks allows us to accumulate a knowledge base that can dramatically facilitate the construction of new translation tasks. The experimental results shows the flexibility of the proposed approach, which makes it possible to deal with realistic face translation tasks.

Chapter 7

Deep Logic Models

Blending symbolic and sub-symbolic techniques is one of the most challenging open problem in AI and, recently, a lot of works, often referred as neuro-symbolic approaches [45], have been proposed by several authors [16, 64, 95, 124,125]. In previous chapters, we already discussed some of the most studied frameworks combining machine learning techniques and logical inference, and in Ch. 6 we presented a new system called LYRICS, providing an interface to integrate both of these approaches. However, all these frameworks present some drawbacks and limitations. For instance, Probabilistic Soft Logic builds an undirected graphical model to represent a grounded FOL knowledge base and, employing a differentiable approximation of FOL, it allows to learn the weight of each formula in the KB by maximizing the log–likelihood of the training data. However, PSL focuses on logic reasoning without any solid integration with deep learners, beside a simple stacking with no joint training. On the other hand, systems like Semantic–Based Regularization, Logic Tensor Network, or LYRICS overcome this limitation exploiting kernel machines or other machine learning architectures to learn regularities from data. All these frameworks share the same basic idea of integrating logical reasoning and learning using a continuous relaxation of logic. However, this class of approaches considers the reasoning layer as frozen, without allowing to jointly train its parameters. This is a big limitation, as these methods work better only with hard constraints, while they are less suitable in presence of reasoning under uncertainty.

In this chapter, we present a new class of models allowing to overcome the main drawbacks of existing approaches that are called *Deep Logic Models* (DLMs) [97], which is recently developed and an ongoing study. DLMs provide a unified framework to integrate probabilistic logic reasoning and deep learning, exploiting both an input layer processing the sensorial input patterns and a higher level which enforces some structure to the model output. Unlike in Semantic–based Regularization [34] or Logic Tensor Networks [38], the sensorial and reasoning layers can be jointly trained, so that the high-level weights imposing the output structure are jointly learned together with the neural network weights, processing the low-level input. The bonding is very general as any (set of) deep learners can be integrated and any output structure can be expressed.

Definition of the Model

We indicate as $\boldsymbol{\theta}$ the model parameters, and $\mathcal{X}$ the collection of input sensorial data. Deep Logic Models (DLMs) assume that the prediction of the system is constrained by the available prior knowledge. Therefore, unlike standard neural networks which compute the output via a simple forward pass, the output computation in DLM can be decomposed into two stages: a *low-level* stage processing the input patterns, and a subsequent *semantic* stage, expressing constraints over the output and performing higher level reasoning. We indicate by $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ and by $\boldsymbol{f} = \{f_1, \ldots, f_n\}$ the two multivariate random variables corresponding to the output of the model and to the output of the first stage respectively, where $n > 0$ denotes the dimension of the model outcomes. Assuming that the input data is processed using neural networks, the model parameters can be split into two independent components $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\lambda}\}$, where $\boldsymbol{w}$ is the vector of weights of the networks f_{nn} and $\boldsymbol{\lambda}$ is the vector of weights of the second stage, controlling the semantic layer and the constraint enforcement. Figure 7.1 shows the graphical dependencies among the stochastic variables that are involved in our model. The first layer processes the inputs returning the values $\boldsymbol{f}$ using a model with parameters $\boldsymbol{w}$. The higher layer takes as input $\boldsymbol{f}$ and applies reasoning using a set of constraints, whose parameters are indicated as $\boldsymbol{\lambda}$, then it returns the set of output variables $\boldsymbol{y}$.

The Bayes rule allows to link the probability of the parameters to the posterior and prior distributions:

$$p(\boldsymbol{\theta}|\boldsymbol{y}, \mathcal{X}) \propto p(\boldsymbol{y}|\boldsymbol{\theta}, \mathcal{X}) p(\boldsymbol{\theta}) \ .$$

Assuming the prior may be decomposed into a sensorial and a semantic level,

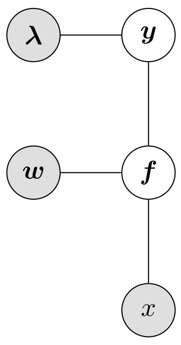

Figure 7.1: *The DLM graphical model assumes that the output variables $\boldsymbol{y}$ depend on the output of first stage $\boldsymbol{f}$, processing the input $\mathcal{X}$. This corresponds to the breakdown into a lower sensorial layer and a high level semantic one.*

namely $p(\boldsymbol{\theta}) = p(\boldsymbol{\lambda})p(\boldsymbol{w})$, and that the posterior can be marginalized over the assignments for $\boldsymbol{f}$ and approximated to the outputs of the neural architectures:

$$p(\boldsymbol{y}|\boldsymbol{\theta}, \mathcal{X}) \approx p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda}) \ .$$

where $\boldsymbol{f}_{nn}$ denotes the actual output of the networks f_{nn} over the inputs.

Definition 7.1 (Deep Logic Models). *A Deep Logic Model assumes that $p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda})$ is modeled via an undirected probabilistic graphical model in the exponential family, such that:*

$$p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda}) = \frac{1}{Z(\boldsymbol{\theta})} \exp\left(\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \Phi_c(\boldsymbol{y})\right) \ , \tag{7.1}$$

where Φ_r and Φ_c are two potential functions expressing the closeness between the output of DLM and the output of the neural networks, and the satisfaction of the c–th logical constraint. The partition function Z depends on both the neural network parameters $\boldsymbol{w}$ by means of $\boldsymbol{f}_{nn}$ and the constraint weights λ_c, and is defined as follows:

$$Z(\boldsymbol{\theta}) = \int_{\boldsymbol{y}} \exp\left(\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \Phi_c(\boldsymbol{y})\right) d\boldsymbol{y} \ .$$

MAP Inference

MAP inference assumes that the model parameters are known and it aims at finding the assignment maximizing $p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda})$. MAP inference does not require to compute the partition function Z which acts as a constant when the weights are fixed. Therefore:

$$\boldsymbol{y}_M = \underset{\boldsymbol{y}}{\operatorname{argmax}} \log p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda}) = \underset{\boldsymbol{y}}{\operatorname{argmax}} \left[\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \Phi_c(\boldsymbol{y})\right] .$$

The above maximization problem can be optimized via gradient descent by computing:

$$\nabla_{\boldsymbol{y}} \log p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda}) = \nabla_{\boldsymbol{y}} \Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \nabla_{\boldsymbol{y}} \Phi_c(\boldsymbol{y})$$

Learning

Training can be carried out by maximizing the likelihood of the training data:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \log p(\boldsymbol{\theta}|\boldsymbol{y}_t, \mathcal{X}) = \log p(\boldsymbol{y}_t|\boldsymbol{\theta}, \mathcal{X}) + \log p(\boldsymbol{w}) + \log p(\boldsymbol{\lambda}) .$$

In particular, assuming that $p(\boldsymbol{y}_t|\boldsymbol{\theta}, \mathcal{X})$ follows the model defined in equation (7.1) and the parameter priors follow Gaussian distributions, we get:

$$\log p(\boldsymbol{\theta}|\boldsymbol{y}_t, \mathcal{X}) = -\frac{\alpha}{2}||\boldsymbol{w}||^2 - \frac{\beta}{2}||\boldsymbol{\lambda}||^2 - \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \Phi_c(\boldsymbol{y}_t) - \log Z(\boldsymbol{\theta})$$

where α, β are meta-parameters determined by the variance of the selected Gaussian distributions. Also in this case the likelihood may be maximized by gradient descent using the following derivatives with respect to the model parameters:

$$\begin{aligned} \frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{y}_t, \mathcal{X})}{\partial \lambda_c} &= -\beta\lambda_c + \Phi_c(\boldsymbol{y}_t) - E_p\left[\Phi_c\right] \\ \frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{y}_t, \mathcal{X})}{\partial w_i} &= -\alpha w_i + \frac{\partial \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn})}{\partial w_i} - E_p\left[\frac{\partial \Phi_r}{\partial w_i}\right] \end{aligned}$$

Unfortunately, the direct computation of the expected values in the above derivatives is not feasible. A possible approximation [55, 63] relies on replacing the expected values with the corresponding value at the MAP solution, assuming that most of the probability mass of the distribution is centered around it. This can be done directly on the above expressions for the derivatives or in the log likelihood:

$$\log p(\boldsymbol{y}_t|\boldsymbol{f}_{nn}, \mathcal{X}) \approx \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) - \Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \left(\Phi_c(\boldsymbol{y}_t) - \Phi_c(\boldsymbol{y}_M)\right)$$

From the above approximation, it emerges that the likelihood tends to be maximized when the MAP solution is close to the training data, namely if $\Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) \simeq \Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn})$ and $\Phi_c(\boldsymbol{y}_t) \simeq \Phi_c(\boldsymbol{y}_M)$ $\forall c$. Furthermore, the probability distribution is more centered around the MAP solution when $\Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn})$ is close to its maximum value. We assume that Φ_r is negative and have zero as upper bound: $\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) \leq 0$ $\forall \boldsymbol{y}, \boldsymbol{f}_{nn}$, like it holds for the already mentioned negative quadratic potential $\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) = -\frac{1}{2}||\boldsymbol{y} - \boldsymbol{f}_{nn}||^2$. Therefore, the constraint $\Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) \simeq \Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn})$ is transformed into the two separate constraints $\Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) \simeq 0$ and $\Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn}) \simeq 0$.

This means that, given the current MAP solution, it is possible to increase the log likelihood by computing the gradient and weight updates using the following cost function:

$$\log p(\boldsymbol{w}) + \log p(\boldsymbol{\lambda}) + \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) + \Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \left[\Phi_c(\boldsymbol{y}_t) - \Phi_c(\boldsymbol{y}_M)\right]$$

In this paper, a quadratic form for the priors and the potentials is selected, but other choices are possible. For example, $\Phi_r(\cdot)$ could instead be implemented as a negative cross–entropy loss. Therefore, replacing the selected forms for the potentials and changing the sign to transform a maximization into a minimization problem, yields the following cost function, given the current MAP solution:

$$\begin{aligned} C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, \mathcal{X}) &= \frac{\alpha}{2}||\boldsymbol{w}||^2 + \frac{\beta}{2}||\boldsymbol{\lambda}||^2 + \frac{1}{2}||\boldsymbol{y}_t - \boldsymbol{f}_{nn}||^2 + \frac{1}{2}||\boldsymbol{y}_M - \boldsymbol{f}_{nn}||^2 + \\ &+ \sum_c \lambda_c \left[\Phi_c(\boldsymbol{y}_t) - \Phi_c(\boldsymbol{y}_M)\right] \ . \end{aligned}$$

Minimizing the cost function $C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, \mathcal{X})$ is just a local approximation of the full likelihood maximization for the current MAP solution. Therefore, the training process alternates the computation of the MAP solution, the computation of the gradient for $C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, \mathcal{X})$ and one weight update step.

Please note that, for any constraint c, the parameter λ_c admits also a negative value. This is in case the c–th constraint turns out to be too satisfied by the actual MAP solution with respect to the satisfaction degree on the training data.

Chapter 8

Conclusions and Future Work

This chapter summarizes the work presented in this thesis, by enlightening both the main contributions of the research and some related consequences. In addition, some possible future directions are discussed as well as some promising development of the presented framework.

8.1 Concluding Remarks

In the following, we discuss some aspects of the main results we provided as well as some related issues.

Mapping Logical Formulas into Loss Functions

In this thesis we proposed a theoretical framework that allows us to convert first–order logic formulas into functional constraints that can be embedded into the training procedure for a learning agent. In this respect, the main contribution of this work is the characterization of the fragment of Łukasiewicz logic yielding convex constraints. This result may be exploited in several learning schemata where FOL formulas provide prior-knowledge and in particular, we discuss the case of kernel machines and collective classification, together with some experimental analysis. Indeed in principle, whenever we are given a set of logical formulas, we can translate them into an equivalent form with only conjunctions, disjunctions and negations on propositional variables. Then if we translate them by the convex fragment of **Ł**, the constraints turn out to be convex and also equivalent to a set of linear constraints. Convexity in general allows the definition of more efficient methods for the optimization process at the basis of the learning process and we show how both kernel machines and collective classification can be formulated by quadratic programming.

A deeper analysis relates the mapping of t-norm fuzzy logic functional representation of formulas to the generator function of the considered t-norm (if it has any). This approach may be applied in case of generated archimedean t-norms, e.g. Łukasiewicz and Product logics and we show some examples of how formulas may be mapped to functional constraints exploiting the additive generator function. This allows us to recover classic machine learning loss functions (e.g. the cross–entropy loss) and also to produce a wide class of new ones by varying the generator functions.

Unnecessary Logical Constraints

In general, in learning from constraints, several constraints are combined into an optimization scheme and often it is quite difficult to identify the contribution of each of them. In particular, some constraints could turn out to be not necessary for the solution. In this thesis, we propose a formal definition of unnecessary constraint as well as a method to determine which are the unnecessary constraints for a learning process in a multi-task problem. A crucial role for the necessity of a certain constraint is played by consequence relations among the constraints that are enforced at the same time. This is a reason why logical constraints are suitable for this kind of investigation. At hand, they are quite general to include both pointwise and consistency constraints. In addition, we can exploit all the studies on truth preservation for formulas. In particular, the logical consequence among formulas is a sufficient condition to conclude that a constraint, corresponding to a certain formula, is unnecessary. However, we also provide an algebraic condition that turns out to be both necessary and sufficient in case the problem is strictly-convex.

LYRICS

This thesis also presents a novel and general framework, called LYRICS, to bridge logic reasoning and deep learning. The framework is directly implemented in TensorFlow, allowing a seaming-less integration that is architecture agnostic. The frontend of the framework is a declarative language based on first–order logic. In particular, we present a set of examples illustrating the generality and expressiveness of the framework that can be applied to a large range of tasks, including classification, pattern generation and symbolic reasoning. However, a special mention is reserved to visual generation tasks. A distinguishing property of the approach is that the description of generation problems using logic is very flexible and it allows us to tackle very different translation problems with little effort. Instead of looking for specific handcrafted cost functions, the proposed approach offers a general scheme for their construction that arises from the t-norm theory. Moreover, the interleaving of different image translations tasks allows us to accumulate a knowledge base that can facilitate the construction of new translation tasks. The experimental results show the flexibility of the proposed approach, which makes it possible to obtain very realistic images in the considered face translation tasks.

8.2 Future Work

In the following, we give an insight of possible future works. Such arguments involve both the results presented in this thesis and a novel model we are working on that has already sketched in Ch. 7.

Convex Fragment In principle any boolean FOL formula may be converted according to the convex Łukasiewicz fragment into a convex function. However, as we already noticed in Sec. 4.1.3, this conversion can affect the intuitive meaning associated to the initial formula. For instance, in Example 4.5 it is discussed the case of a mutual exclusion rule, namely we are interested in forcing that a certain pattern belongs to only one of two given classes. We recall that in Łukasiewicz logic, we are given two disjunctions, the weak $x \vee y = \max\{x, y\}$ and the strong $x \oplus y = \min\{1, x + y\}$. This latter is in the fragment yielding convex constraints while the former is not. However in this case, the weak disjunction seems to better approximate the formula intent, indeed it is satisfied if and only if $x = 1$ or $y = 1$, whereas the strong one it is satisfied whenever $x + y \geq 1$.

Lines of research.

1. Considering a continuation method for different conversion options of a

certain formula. For instance, for mutual exclusion we can consider

$$\lambda \cdot \max\{x, y\} + (1 - \lambda) \cdot \min\{1, x + y\} \ ,$$

where $\lambda \in [0, 1]$ is a combination parameter that controls the deviation from the convex solution.

2. Extending the presented results to other contexts beside those already considered, e.g. *reinforcement learning*, where logical rules could be exploited to define the reward function, as for instance considered in [88].

Loss and Generators Given a fuzzy representation of formulas by a continuous archimedean t-norm T, we described some consequences of mapping into functional constraints by an additive generator of T. However in principle, one may think to introduce a new parameter in the loss function by considering a parametric generator function, as introduced in Sec. 2.3.1.

Lines of research.

1. A promising idea we plan to investigate is the possibility to jointly learn the loss function, once it is expressed by means of a t-norm parametrized generator function.

In this view, one may think to learn or validate (e.g. by cross–validation) also the loss function that is responsible of the learning process. Depending on a certain parameter, we can determine the most effective loss function with respect to a certain learning problem according to the available data.

Support Constraints In Sec. 5.2.2, we provide a formal definition of unnecessary constraint for both the hard and soft case. However, we did not prove any specific result for the soft-constraint case, indeed, the setting with soft-constraints also depends on the presence of slack values and it requires more investigations. In addition, for what concerns logical deduction, we should consider different logical arguments to deal with intermediate degree of consequence.

Lines of research.

1. We plan to extend the algebraic results in propositions 5.2 and 5.3 to unnecessary soft-constraints.
2. Consider other fuzzy logic consequence relations, such as the degree preservation among arbitrary $[0, 1]$–values in formulas. Indeed, this kind of consequence allows us to consider the truth preservation also for logical constraints with not null slack variables.

Deep Logic Models In order to develop a new framework allowing to improve existing approaches integrating probabilistic logic reasoning and deep learning, in Ch. 7 we briefly introduce a new class of models, called *Deep Logic Models* (DLMs). DLMs mainly focus on expressing the high-level structure using logic formalism like first–order logic (FOL). In particular, a consistent and fully differentiable relaxation of FOL is used to map the knowledge into a set of potentials that can be used in training and inference. Moreover, DLMs allow a relation to be encoded by any selected function (e.g. any deep neural networks), which is co-trained during learning. Therefore, DLMs are capable of a more powerful and flexible exploitation of the representation space.

Lines of research.

1. So far, Deep Logic Models have been validated only on some toy examples. For future works we plan to extend the experimental analysis to more complex tasks in order to validate and improve the methodology.
2. Deep Logic Models also open up the possibility to iteratively integrate rule induction mechanisms like the ones proposed by the Inductive Logic Programming community [85, 116]. We plan to include in the learning process a mechanism to infer new logical statements from examples.

List of Publications

Below a list of the papers produced during the period of this research is reported.

Journal Articles

1. Francesco Giannini, Michelangelo Diligenti, Marco Gori, and Marco Maggini. On a convex logic fragment for learning and reasoning. *IEEE Transactions on Fuzzy Systems*, 2018.

Conference Articles

1. Francesco Giannini, Michelangelo Diligenti, Marco Gori, and Marco Maggini. Learning łukasiewicz logic fragments by quadratic programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 410–426. Springer, 2017

2. Francesco Giannini, Michelangelo Diligenti, Marco Gori, and Marco Maggini. Characterization of the convex łukasiewicz fragment for learning from constraints. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Technical Reports

1. Francesco Giannini, Vincenzo Laveglia, Alessandro Rossi, Dario Zanca, and Andrea Zugarini. Neural Networks for Beginners. A fast implementation in Matlab, Torch, TensorFlow. arXiv preprint arXiv: 1703.05298, 2017.

Arxiv Preprints of Submitted Articles

1. Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Constraint-based visual generation. *arXiv preprint arXiv: 1807.09202*, 2018.

2. Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. Integrating learning and reasoning with deep logic models. *arXiv preprint arXiv:1901.04195*, 2019.

.

Bibliography

[1] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)

[2] Aguiló, I., Carbonell, M., Suñer, J., Torrens, J.: Dual representable aggregation functions and their derived s-implications. In: International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems. pp. 408–417. Springer (2010)

[3] Aguzzoli, S., Bova, S., Gerla, B.: Chapter ix: Free algebras and functional representation for fuzzy logics (2011)

[4] Alsina, C., Trillas, E., Valverde, L.: On some logical connectives for fuzzy sets theory. Journal of Mathematical Analysis and Applications 93(1), 15–26 (1983)

[5] Aronszajn, N.: Theory of reproducing kernels. Transactions of the American mathematical society 68(3), 337–404 (1950)

[6] Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. arXiv preprint arXiv:1505.04406 (2015)

[7] Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. Journal of Machine Learning Research 18, 1–67 (2017)

[8] Bach, S.H., Huang, B., London, B., Getoor, L.: Hinge-loss markov random fields: convex inference for structured prediction. In: Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence. pp. 32–41. AUAI Press (2013)

[9] Bauml, M., Tapaswi, M., Stiefelhagen, R.: Semi-supervised learning with constraints for person identification in multimedia data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3602–3609 (2013)

[10] Beliakov, G., Pradera, A., Calvo, T.: Aggregation functions: A guide for practitioners, vol. 221. Springer (2007)

[11] Bofill, M., Manya, F., Vidal, A., Villaret, M.: Finding hard instances of satisfiability in lukasiewicz logics. In: Multiple-Valued Logic (ISMVL), 2015 IEEE International Symposium on. pp. 30–35. IEEE (2015)

[12] Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory. pp. 144–152. ACM (1992)

[13] Bröcheler, M., Mihalkova, L., Getoor, L.: Probabilistic similarity logic. In: Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence. pp. 73–82. AUAI Press (2010)

[14] Bullen, P.S.: Handbook of means and their inequalities, vol. 560. Springer Science & Business Media (2013)

[15] Calvo, T., Kolesárová, A., Komorníková, M., Mesiar, R.: Aggregation operators: properties, classes and construction methods. In: Aggregation operators, pp. 3–104. Springer (2002)

[16] Chen, L.C., Schwing, A., Yuille, A., Urtasun, R.: Learning deep structured models. In: International Conference on Machine Learning. pp. 1785–1794 (2015)

[17] Cintula, P., Klement, E.P., Mesiar, R., Navara, M.: Fuzzy logics with an additional involutive negation. Fuzzy Sets and Systems 161(3), 390–411 (2010)

[18] Clocksin, W.F., Mellish, C.S.: Programming in Prolog: Using the ISO standard. Springer Science & Business Media (2012)

[19] Cohen, W.W.: Tensorlog: A differentiable deductive database. arXiv preprint arXiv:1605.06523 (2016)

[20] Cortes, C., Vapnik, V.: Support-vector networks. Machine Learning 20(3), 273–297 (Sep 1995), `https://doi.org/10.1023/A:1022627411411`

[21] Cortes, C., Vapnik, V.: Support-vector networks. Machine learning 20(3), 273–297 (1995)

[22] Cumby, C.M., Roth, D.: On kernel methods for relational learning. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 107–114 (2003)

[23] Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems 2(4), 303–314 (1989)

[24] De Baets, B., Fodor, J.: Residual operators of uninorms. Soft Computing 3(2), 89–100 (1999)

[25] De Raedt, L., Dries, A., Guns, T., Bessiere, C.: Learning constraint satisfaction problems: An ilp perspective. In: Data Mining and Constraint Programming, pp. 96–112. Springer (2016)

[26] De Raedt, L., Kersting, K.: Probabilistic logic learning. ACM SIGKDD Explorations Newsletter 5(1), 31–48 (2003)

[27] De Raedt, L., Kersting, K.: Probabilistic inductive logic programming. In: Probabilistic Inductive Logic Programming, pp. 1–27. Springer (2008)

[28] De Raedt, L., Kimmig, A.: Probabilistic (logic) programming concepts. Machine Learning 100(1), 5–47 (2015)

[29] De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 2468–2473. IJCAI'07, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007), http://dl.acm.org/citation.cfm?id=1625275.1625673

[30] De Raedt, L., Passerini, A., Teso, S.: Learning constraints from examples. In: Proceedings in Thirty-Second AAAI Conference on Artificial Intelligence, AAAI, New Orleans, USA. pp. 02–07 (2018)

[31] De Una, D., Rümmele, N., Gange, G., Schachte, P., Stuckey, P.J.: Machine learning and constraint programming for relational-to-ontology schema mapping. In: IJCAI. pp. 1277–1283 (2018)

[32] Demeester, T., Rocktäschel, T., Riedel, S.: Lifted rule injection for relation embeddings. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 1389–1399 (2016)

[33] Diligenti, M., Gori, M., Maggini, M., Rigutini, L.: Bridging logic and kernel machines. Machine learning 86(1), 57–88 (2012)

[34] Diligenti, M., Gori, M., Sacca, C.: Semantic-based regularization for learning and inference. Artificial Intelligence 244, 143–165 (2017)

[35] Diligenti, M., Gori, M., Scoca, V.: Learning efficiently in semantic based regularization. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 33–46. Springer (2016)

[36] Diligenti, M., Roychowdhury, S., Gori, M.: Image classification using deep learning and prior knowledge. In: Proceedings of Third International Workshop on Declarative Learning Based Programming (DeLBP) (February 2018)

[37] Dillon, J.V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., Saurous, R.A.: Tensorflow distributions. arXiv preprint arXiv:1711.10604 (2017)

[38] Donadello, I., Serafini, L., d'Avila Garcez, A.: Logic tensor networks for semantic image interpretation. In: IJCAI International Joint Conference on Artificial Intelligence. pp. 1596–1602 (2017)

[39] Ebbinghaus, H.D., Flum, J., Thomas, W.: Mathematical logic. Springer Science & Business Media (2013)

[40] Enderton, H., Enderton, H.B.: A mathematical introduction to logic. Elsevier (2001)

[41] Esteva, F., Godo, L., Hájek, P., Navara, M.: Residuated fuzzy logics with an involutive negation. Archive for mathematical logic 39(2), 103–124 (2000)

[42] Fierens, D., Van den Broeck, G., Renkens, J., Shterionov, D., Gutmann, B., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted boolean formulas. Theory and Practice of Logic Programming 15(3), 358–401 (2015)

[43] Flaminio, T., Marchioni, E.: T-norm-based logics with an independent involutive negation. Fuzzy Sets and Systems 157(24), 3125–3144 (2006)

[44] Frandina, S., Sacca, C., Diligenti, M., Gori, M.: Constraint-based learning for text categorization. In: WORKSHOP ON COMBINING. Citeseer (2012)

[45] Garcez, A.S.d., Broda, K.B., Gabbay, D.M.: Neural-symbolic learning systems: foundations and applications. Springer Science & Business Media (2012)

[46] Gärtner, T.: A survey of kernels for structured data. ACM SIGKDD Explorations Newsletter 5(1), 49–58 (2003)

[47] Gehrke, M., Walker, C., Walker, E.: Demorgan systems on the unit interval. International Journal of Intelligent Systems 11(10), 733–750 (1996)

[48] Gehrke, M., Walker, C., Walker, E.: Fuzzy logics arising from strict de morgan systems. In: Topological and Algebraic Structures in Fuzzy Sets, pp. 257–276. Springer (2003)

[49] Getoor, L., Taskar, B.: Introduction to statistical relational learning, vol. 1. MIT press Cambridge (2007)

[50] Giannini, F., Diligenti, M., Gori, M., Maggini, M.: Learning łukasiewicz logic fragments by quadratic programming. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 410–426. Springer (2017)

[51] Giannini, F., Diligenti, M., Gori, M., Maggini, M.: Characterization of the convex łukasiewicz fragment for learning from constraints. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

[52] Giannini, F., Diligenti, M., Gori, M., Maggini, M.: On a convex logic fragment for learning and reasoning. IEEE Transactions on Fuzzy Systems (2018)

[53] Gnecco, G., Gori, M., Melacci, S., Sanguineti, M.: Foundations of support constraint machines. Neural computation 27(2), 388–480 (2015)

[54] Gnecco, G., Gori, M., Melacci, S., Sanguineti, M.: Learning with mixed hard/-soft pointwise constraints. IEEE transactions on neural networks and learning systems 26(9), 2019–2032 (2015)

[55] Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)

[56] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in neural information processing systems. pp. 2672–2680 (2014)

[57] Gori, M.: Machine Learning: A Constraint-based Approach. Morgan Kaufmann (2017)

[58] Gori, M., Melacci, S.: Support constraint machines. In: Lu, B.L., Zhang, L., Kwok, J. (eds.) Neural Information Processing. pp. 28–37. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)

[59] Gori, M., Melacci, S.: Constraint verification with kernel machines. IEEE transactions on neural networks and learning systems 24(5), 825–831 (2013)

[60] Grabisch, M., Marichal, J.L., Mesiar, R., Pap, E.: Aggregation functions: means. Information Sciences 181(1), 1–22 (2011)

[61] Gutiérrez-Basulto, V., Jung, J.C., Kuzelka, O.: Quantified markov logic networks. In: KR (2018)

[62] Hájek, P.: Metamathematics of fuzzy logic, vol. 4. Springer Science & Business Media (2013)

[63] Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edn. (1994)

[64] Hazan, T., Schwing, A.G., Urtasun, R.: Blending learning and inference in conditional random fields. The Journal of Machine Learning Research 17(1), 8305–8329 (2016)

[65] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

[66] Hu, Z., Ma, X., Liu, Z., Hovy, E., Xing, E.: Harnessing deep neural networks with logic rules. arXiv preprint arXiv:1603.06318 (2016)

[67] Hwang, J.N., Hu, Y.H.: Handbook of neural network signal processing. CRC press (2001)

[68] Jaffar, J., Maher, M.J.: Constraint logic programming: A survey. The journal of logic programming 19, 503–581 (1994)

[69] Jenei, S.: A note on the ordinal sum theorem and its consequence for the construction of triangular norms. Fuzzy Sets and Systems 126(2), 199–205 (2002)

[70] Jensen, F.V.: An introduction to Bayesian networks, vol. 210. UCL press London (1996)

[71] Jung, J.H., O'leary, D.P., Tits, A.L.: Adaptive constraint reduction for training support vector machines. Electronic Transactions on Numerical Analysis 31, 156–177 (2008)

[72] Jung, J.H., O'Leary, D.P., Tits, A.L.: Adaptive constraint reduction for convex quadratic programming. Computational Optimization and Applications 51(1), 125–157 (2012)

[73] Kathryn, W.W.C.F.Y., Mazaitis, R.: Tensorlog: Deep learning meets probabilistic databases. Journal of Artificial Intelligence Research 1, 1–15 (2018)

[74] Khosravi, H., Bina, B.: A survey on statistical relational learning. In: Canadian Conference on Artificial Intelligence. pp. 256–268. Springer (2010)

[75] Kimmig, A., Bach, S., Broecheler, M., Huang, B., Getoor, L.: A short introduction to probabilistic soft logic. In: Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications. pp. 1–4 (2012)

[76] Kimmig, A., Van den Broeck, G., De Raedt, L.: An algebraic prolog for reasoning about possible worlds. In: AAAI (2011)

[77] Kindermann, R.: Markov random fields and their applications. American mathematical society (1980)

[78] Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. position paper i: basic analytical and algebraic properties. Fuzzy Sets and Systems 143(1), 5–26 (2004)

[79] Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. position paper ii: general constructions and parameterized families. Fuzzy Sets and Systems 145(3), 411–438 (2004)

[80] Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. position paper iii: continuous t-norms. Fuzzy Sets and Systems 145(3), 439–454 (2004)

[81] Klement, E.P., Mesiar, R., Pap, E.: Triangular norms, vol. 8. Springer Science & Business Media (2013)

[82] Kok, S., Domingos, P.: Learning the structure of markov logic networks. In: Proceedings of the 22nd international conference on Machine learning. pp. 441–448. ACM (2005)

[83] Kolb, S., Teso, S., Passerini, A., De Raedt, L.: Learning smt (lra) constraints using smt solvers. In: IJCAI. pp. 2333–2340 (2018)

[84] Landwehr, N., Passerini, A., De Raedt, L., Frasconi, P.: Fast learning of relational kernels. Machine learning 78(3), 305–342 (2010)

[85] Lavrac, N., Dzeroski, S.: Inductive logic programming. In: WLP. pp. 146–160. Springer (1994)

[86] Lawrence, S., Giles, C.L., Tsoi, A.C., Back, A.D.: Face recognition: A convolutional neural-network approach. IEEE transactions on neural networks 8(1), 98–113 (1997)

[87] Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., Carin, L.: Alice: Towards understanding adversarial learning for joint distribution matching. In: Advances in Neural Information Processing Systems. pp. 5501–5509 (2017)

[88] Li, X., Vasile, C.I., Belta, C.: Reinforcement learning with temporal logic rewards. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 3834–3839. IEEE (2017)

[89] Liu, H.W.: Semi-uninorms and implications on a complete lattice. Fuzzy Sets and Systems 191, 72–82 (2012)

[90] Liu, M.Y., Breuel, T., Kautz, J.: Unsupervised image-to-image translation networks. In: Advances in Neural Information Processing Systems. pp. 700–708 (2017)

[91] Liu, M.Y., Tuzel, O.: Coupled generative adversarial networks. In: Proceedings of the 30th International Conference on Neural Information Processing Systems. pp. 469–477. Curran Associates Inc. (2016)

[92] Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV) (December 2015)

[93] Lowe, D.G.: Object recognition from local scale-invariant features. In: Computer vision, 1999. The proceedings of the seventh IEEE international conference on. vol. 2, pp. 1150–1157. IEEE (1999)

[94] Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5188–5196 (2015)

[95] Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., De Raedt, L.: Deepproblog: Neural probabilistic logic programming. arXiv preprint arXiv:1805.10872 (2018)

[96] Marchioni, E., Wooldridge, M.: Łukasiewicz games: A logic-based approach to quantitative strategic interactions. ACM Transactions on Computational Logic (TOCL) 16(4), 33 (2015)

[97] Marra, G., Giannini, F., Diligenti, M., Gori, M.: Integrating learning and reasoning with deep logic models. arXiv preprint arXiv:1901.04195 (2019)

[98] Marra, G., Giannini, F., Diligenti, M., Gori, M.: Lyrics: a general interface layer to integrate ai and deep learning. arXiv preprint arXiv:1903.07534 (2019)

[99] Mas, M., Monserrat, M., Torrens, J.: A characterization of (u, n), ru, ql and d-implications derived from uninorms satisfying the law of importation. Fuzzy Sets and Systems 161(10), 1369–1387 (2010)

[100] Mas, M., Monserrat, M., Torrens, J., Trillas, E.: A survey on fuzzy implication functions. IEEE Transactions on fuzzy systems 15(6), 1107–1121 (2007)

[101] Mayoh, B., Tyugu, E., Penjam, J.: Constraint programming, vol. 131. Springer Science & Business Media (2013)

[102] Minervini, P., Demeester, T., Rocktäschel, T., Riedel, S.: Adversarial sets for regularising neural link predictors. arXiv preprint arXiv:1707.07596 (2017)

[103] Minsky, M., Papert, S.A.: Perceptrons: An introduction to computational geometry. MIT press (2017)

[104] Møller, M.F.: A scaled conjugate gradient algorithm for fast supervised learning. Neural networks 6(4), 525–533 (1993)

[105] Mosca, A., Magoulas, G.D.: Adapting resilient propagation for deep learning. CoRR abs/1509.04612 (2015), `http://arxiv.org/abs/1509.04612`

[106] Mostert, P.S., Shields, A.L.: On the structure of semigroups on a compact manifold with boundary. Annals of Mathematics pp. 117–143 (1957)

[107] Muggleton, S.: Inductive logic programming. New generation computing 8(4), 295–318 (1991)

[108] Muggleton, S.: Bayesian inductive logic programming. In: Proceedings of the seventh annual conference on Computational learning theory. pp. 3–11. ACM (1994)

[109] Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming 19, 629–679 (1994)

[110] Muggleton, S., Lodhi, H., Amini, A., Sternberg, M.J.: Support vector inductive logic programming. In: Discovery science. vol. 3735, pp. 163–175. Springer (2005)

[111] Muggleton, S., et al.: Stochastic logic programs. Advances in inductive logic programming 32, 254–264 (1996)

[112] Nasrabadi, N.M.: Pattern recognition and machine learning. Journal of electronic imaging 16(4), 049901 (2007)

[113] Neyshabur, B., Tomioka, R., Srebro, N.: Norm-based capacity control in neural networks. In: Conference on Learning Theory. pp. 1376–1401 (2015)

[114] Novák, V., Perfilieva, I., Mockor, J.: Mathematical principles of fuzzy logic, vol. 517. Springer Science & Business Media (2012)

[115] Pradera, A., Beliakov, G., Bustince, H., De Baets, B.: A review of the relationships between implication, negation and aggregation functions from the point of view of material implication. Information Sciences 329, 357–380 (2016)

[116] Quinlan, J.R.: Learning logical definitions from relations. Machine learning 5(3), 239–266 (1990)

[117] Raedt, L.D., Kersting, K., Natarajan, S., Poole, D.: Statistical relational artificial intelligence: Logic, probability, and computation. Synthesis Lectures on Artificial Intelligence and Machine Learning 10(2), 1–189 (2016)

[118] Ramík, J., Vlach, M.: Aggregation functions and generalized convexity in fuzzy optimization and decision making. Annals of Operations Research 195(1), 261–276 (2012)

[119] Ravi, S.N., Dinh, T., Lokhande, V.S.R., Singh, V.: Constrained deep learning using conditional gradient and applications in computer vision. arXiv preprint arXiv:1803.06453 (2018)

[120] Richardson, M., Domingos, P.: Markov logic networks. Machine learning 62(1), 107–136 (2006)

[121] Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The rprop algorithm. In: IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS. pp. 586–591 (1993)

[122] Robert, C.: Machine learning, a probabilistic perspective (2014)

[123] Rockafellar, R.T., Wets, R.J.B.: Variational analysis, vol. 317. Springer Science & Business Media (2009)

[124] Rocktäschel, T., Riedel, S.: Learning knowledge base inference with neural theorem provers. In: Proceedings of the 5th Workshop on Automated Knowledge Base Construction. pp. 45–50 (2016)

[125] Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Advances in Neural Information Processing Systems. pp. 3788–3800 (2017)

[126] Rocktäschel, T., Singh, S., Riedel, S.: Injecting logical background knowledge into embeddings for relation extraction. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 1119–1129 (2015)

[127] Rosca, M., Lakshminarayanan, B., Mohamed, D.W.F.S.: Variational approaches for auto-encoding generative adversarial networks. stat 1050, 15 (2017)

[128] Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review 65(6), 386 (1958)

[129] Rossi, F., Van Beek, P., Walsh, T.: Handbook of constraint programming. Elsevier (2006)

[130] Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. nature 323(6088), 533 (1986)

[131] de Salvo Braz, R., Amir, E., Roth, D.: A survey of first-order probabilistic models. In: Innovations in Bayesian Networks, pp. 289–317. Springer (2008)

[132] Sato, T., Kameya, Y.: Prism: a language for symbolic-statistical modeling. In: IJCAI. vol. 97, pp. 1330–1339 (1997)

[133] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Transactions on Neural Networks 20(1), 61–80 (2009)

[134] Schölkopf, B., Smola, A.J., Bach, F., et al.: Learning with kernels: support vector machines, regularization, optimization, and beyond. MIT press (2002)

[135] Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine 29(3), 93 (2008)

[136] Serafini, L., Garcez, A.d.: Logic tensor networks: Deep learning and logical reasoning from data and knowledge. arXiv preprint arXiv:1606.04422 (2016)

[137] Serafini, L., Garcez, A.S.d.: Learning and reasoning with logic tensor networks. In: AI* IA. pp. 334–348 (2016)

[138] Shoenfield, J.R.: Mathematical logic. AK Peters/CRC Press (2010)

[139] Socher, R., Chen, D., Manning, C.D., Ng, A.: Reasoning with neural tensor networks for knowledge base completion. In: Advances in neural information processing systems. pp. 926–934 (2013)

[140] Sundermeyer, M., Schlüter, R., Ney, H.: Lstm neural networks for language modeling. In: Thirteenth annual conference of the international speech communication association (2012)

[141] Szu, H.H.: Non-convex optimization. In: Real-Time Signal Processing IX. vol. 698, pp. 59–68. International Society for Optics and Photonics (1986)

[142] Tits, A.L., Absil, P.A., Woessner, W.P.: Constraint reduction for linear programs with many inequality constraints. SIAM Journal on Optimization 17(1), 119–146 (2006)

[143] Torra, V., Narukawa, Y.: Modeling decisions: information fusion and aggregation operators. Springer Science & Business Media (2007)

[144] Tran, D., Hoffman, M.D., Saurous, R.A., Brevdo, E., Murphy, K., Blei, D.M.: Deep probabilistic programming. In: International Conference on Learning Representations (2017)

[145] Trillas, E.: Sobre funciones de negación en la teoría de conjuntos difusos. Stochastica 3(1), 47–60 (1979)

[146] Turing, A.M.: Computing machinery and intelligence. In: Parsing the Turing Test, pp. 23–65. Springer (2009)

[147] Wang, W.Y., Cohen, W.W.: Learning first-order logic embeddings via matrix factorization. In: IJCAI. pp. 2132–2138 (2016)

[148] Wang, W.Y., Mazaitis, K., Cohen, W.W.: Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In: Proceedings of the 22nd ACM international conference on Information & Knowledge Management. pp. 2129–2138. ACM (2013)

[149] Wang, Z.: Generating pseudo-t-norms and implication operators. Fuzzy Sets and Systems 157(3), 398–410 (2006)

[150] Williams, C.K., Rasmussen, C.E.: Gaussian processes for regression. In: Advances in neural information processing systems. pp. 514–520 (1996)

[151] Winston, P.H., Horn, B.K.: Lisp. Addison Wesley Pub., Reading, MA (1986)

[152] Wolpert, D.H.: The lack of a priori distinctions between learning algorithms. Neural computation 8(7), 1341–1390 (1996)

[153] Yang, F., Yang, Z., Cohen, W.W.: Differentiable learning of logical rules for knowledge base reasoning. In: Advances in Neural Information Processing Systems. pp. 2319–2328 (2017)

[154] Zadeh, L.A.: Outline of a new approach to the analysis of complex systems and decision processes. IEEE Transactions on systems, Man, and Cybernetics (1), 28–44 (1973)

[155] Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2223–2232 (2017)