UNIVERSITÀ
DEGLI STUDI
FIRENZE

PHD PROGRAM IN SMART COMPUTING
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE (DINFO)

Bridging symbolic and subsymbolic reasoning with MiniMax Entropy models.

Giuseppe Marra

Dissertation presented in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Smart Computing

PhD Program in Smart Computing
University of Florence, University of Pisa, University of Siena

Bridging symbolic and subsymbolic reasoning with MiniMax Entropy models.

Giuseppe Marra

Advisor:

Prof. Marco Gori

Head of the PhD Program:

Prof. Paolo Frasconi

Evaluation Committee:
Prof. Fabrizio Riguzzi, *University of Ferrara*
Prof. Frédéric Precioso, *University Nice Sophia Antipolis*

XXXII ciclo — October 2019

To my family and to my wife

Acknowledgments

This thesis represents only a piece of the larger story of my PhD. A story made of effort and satisfaction, hard work and lot of fun, tears and loud laughters. A story made of travels, dinners (also very spicy ones) and good wines. A story made of many days and many never-ending nights. But, in particular, a story made of wonderful people without whom none of this could have happened.

I would like to thank, first of all, my supervisor Marco Gori, which was the beginning of this story. I owe to him the need to treat science like an art, something I am now addicted to. Searching beauty in science is a way full of great satisfaction.

I would like to thank my Supervisory Committee, composed of Bernardo Magnini and Giovanni Semeraro, whose constant supervision during the entire program allowed me to have an external viewpoint on my work, about its strengths and weaknesses. I thank also my Evaluation Committee, composed of Fabrizio Riguzzi and Frederic Precioso, for the suggestions in improving this thesis.

I would like to thank Luc De Raedt for allowing me to visit his DTAI Laboratory in KU Leuven and all the collegues of the lab. Thanks for allowing me to understand that deep learning is not enough and there is a lot that still need to be solved.

I would like to thank Ondrej Kuzelka for allowing me to visit his IDA Laboratory in CTU Prague and all the collegues of the lab. The interpretation of most of my PhD work in terms of MiniMax Entropy models is due to our collaboration. Talking about AI in a Czech pub drinking Belgian(!?!) beer is something that I suggest to AI practitioners; it can be really inspiring.

I would like to thank all the administrative stuff of both the Siena and Florence universities. In particular, Simona Altamura for her kindness. I would like to thank the Regione Toscana for the Pegaso grant supporting my studies.

A big thank to all the people of the SAILab, students and professors. I will have very good memories about our talks, discussions, dinners and soccer games together! It was really great to be part of our lab! In particular, I thank Michelangelo, for pushing me to the hardest heights (and it is not a metaphor!). I thank Francesco for allowing me to understand that research is made of lots (and lots ... and lots) of endless loud discussions. I thank Andrea and Matteo, my code-mates, for the most funny part of my research: lots of coding and staring at loss green numbers (that sometimes go to 0). I thank Dario for his company in many Sunday lunches and in many other Sundays at the lab; for our jam sessions, for his arancine, for getting always the best bed and also for his unexplainable taste for trash music. I thank all of you for always be by my side.

Finally, I need to thank my family, for teaching me everyday the most difficult art of living. And, above all, I thank Maria Domenica, my wife. Thank you for being the main constant of my life.

Abstract

Deep Learning is considered the foundation of a new technological revolution, which is going to radically change human beings lives. No doubt that, in the following years, the use of deep learning techniques will bring to the development of outstanding applications that will totally change the way people interact with each other, interact with their environment, conceive work, travel, etc.

However, while in the domain of applications deep learning is destined to revolutionize any human activity, it is extremely difficult to believe that these techniques, in an isolated way, can lead to the great dream of general artificial intelligence. Indeed, deep learning algorithms have shown to reach human-like performance in many isolated tasks, like image recognition or speech recognition, but they still struggle in integrating these single activities in a truly intelligent behaviour. Moreover, they are tremendously data-hungry: is recognizing a cat after having seen one million cats an intelligent behaviour?

There is a clear need to look forward for more complex and general theories, where the outstanding deep learning techniques are not a final recipe but only an ingredient. We hope this thesis to be a little step in this direction.

By taking inspiration from both symbolic artificial intelligence, strongly based on mathematical logics, and behavioural sciences, this thesis formulates and investigates a new theory about how intelligent behaviours can be the outcome of the seamlessly integration of two reasoning mechanisms: one subsymbolic, associative and fast; the other symbolic, cautious and slow. This integration naturally emerges from a principle of MiniMax Entropy, where the model of an intelligent system is asked to describe the environment it is exposed to by minimizing its internal confusion, yet keeping the maximum uncertainty about anything it is not able to explain. The theory nicely intercepts multiple AI fields like Statistical Relational Learning, Deep Learning, Constrained Optimization, Probabilistic Graphical Models, Neuro-Symbolic Integration, to name a few.

The theory is extremely general and multiple models can be described in terms of MiniMax Entropy models. Some practical instances of the theory are proposed and empirically investigated, showing competitive results in many different learning and reasoning tasks w.r.t. other specific state-of-the-art approaches. Moreover, actual programming frameworks, inspired by the theory, are proposed and provided to the community for future investigations.

Contents

Chapter 1

Introduction

1.1 Motivations and Goals

The AI revolution. About 100 years ago we did not have wide access to electricity. However, in the last 100 years, the rise of electricity transformed every major industry: communication, travels, entertainment, etc. In the 18th century 70% of American residents were working in agriculture but, today, less than 1% are working in it. This happened in around 200 years only because of the single revolution that happened in the electrical field.

"*Artificial Intelligence is the new electricity*" said Andrew Ng, co-founder of Coursera and Adjunct Professor of Computer Science at Stanford University.

Artificial Intelligence (AI) is predicted to be the next big technological revolution, which is already shaping every aspects of our everyday lives. By a careful inspection, it is easy to see that AI tools are everywhere around us. Vocal assistant in each of our smartphones, controlled ads on the Web, traffic-aware satellite navigators, park-assisted cars, clinical smart files are just few examples of AI in our daily life. But this trend can only increase in the next years. Exactly like electricity, AI will permeate every human activity in an increasingly impactful way. One astonishing statistic describing this trend is the AI market projections in the period 2016-2025 that sees a growth from 1.4 billion $ to 60 billion $.

While the word "AI" is usually the most exploited to collectively describe all these new technologies, experts in the field know that there is one particular class of techniques mainly responsible of this explosion: *deep learning*. Indeed, AI is a field that dates back to the born of computer science itself. However, its impact has only be felt in these last years by the world, thank to the advancement of a class of techniques, called *deep learning*, which evolved original researches in the filed of neural networks. The explosion of deep learning research, together with a huge improvement in distributed computational hardware, made possible the solution of a large class of tasks which, up to few years earlier, were considered human being

prerogative. Examples are image recognition, speech recognition, natural language understanding.

An example of deep learning application that has recently resounded in all media has been the victory of the AlphaGo algorithm against the Go world champion. The Go game, which has always been considered the most difficult board game in existence, has always put computers to the test. An approach that requires evaluating all possible moves should consider a number of cases greater than the number of atoms in the universe. Humans have always been considered superior in that they accompanied a degree of intuition to pure logic. This allows them to proceed by instinct when logic fails.

Informally speaking, the success of deep learning can be attributed to the ability of these algorithms to mimic those human decisions that are more purely instinctive, in the sense that they are not accompanied by a purely conscious reasoning process. Often these are associative tasks, where we respond to sensory input with a decision that is instinctive rather than due to careful and progressive analysis. Consider looking at the image of the cat: you will say that it is a cat without consciously considering the fact that it is a small mammal, with sharp teeth, pointed hilts and long whiskers. Deep learning algorithms try to mimic this: they will say to you that the image is depicting a cat, but they do not carry any logical reasoning behind that decision: it is purely an associative answer. It is worth noticing that it is likely that the success of deep learning should not be attributed necessarily to new and astonishing ideas but mostly to a shift of paradigm. While original AI was more interested in miming those conscious reasoning processes which are usually characteristic of human intelligence, deep learning focused on miming another behaviour of humans, which is connected to instinctive processes.

But: *is deep learning the final recipe to general artificial intelligence?* When talking about general artificial intelligence, we are referring to that set of cognitive abilities that will give machines human-like intelligence as a whole, which is likely to be the big dream of every AI researcher. While there are no doubts that deep learning is making a lot of little steps toward this objective, we believe that there is something still missing, which we are going to motivate now.

Indeed, while discussing about instinctive tasks and decisions, there was a intentional bug in our claim. It is true that a large part of human intelligence and decision making is made of instinctive tasks and that deep learning is increasingly improving in miming these behaviours, but humans are still capable of performing more abstract and conscious reasoning processes on top of these. That is, after having answered about an image to be a cat, the careful human viewer will also try to abstract its answer in a structured way, checking if a logical analysis of its answer make it still sound or not. This analysis (i.e. splitting into parts and then putting everything together in a principled way) is likely to be beneficial for this and other

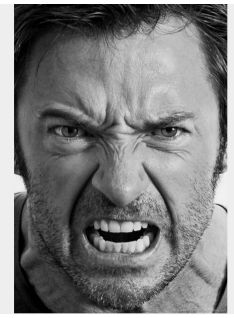
similar tasks in the future. We believe human intelligence to be the never-ending interaction between these two mechanisms: the former that is instinctive, fast, due to many similar experiences, and the latter that is slow and careful, generalizing single experience in general rules.

What this thesis is about. In this thesis, we propose a theory for *modeling* and *integrating* intuitive processes of intelligence, referred to subsymbolic reasoning, and cautious processes of intelligence, referred to symbolic reasoning. Here, deep learning models becomes the building blocks of a more general theory, where objects of the world are related to each other. The resultant relational settings is elegantly described using logical formalism and logical arguments allow for more complex reasoning processes to be carried on. In this way, standard deep learning techniques are extended and complemented by abstract reasoning capabilities.

In the following of this section we want to motivate more deeply the need for such an integration and some of its main features.

Fast and Slow thinking. Talking about human behaviours, this distinction is supported by recent discoveries in the behavioural science, among which those of Daniel Kahneman which made it win the Nobel Prize in Economic Sciences. While going deeper in Kahneman's studies is out of the scope of this work, it is interesting to note that he individuated multiple evidences of this dichotomy in human decision making and general thinking. In particular, a lot of experiments on human behaviours brought him to a model of the human thinking and decision making as the output of the never-ending interaction of two fictitious systems, named System 1 and System 2.

To easily get the main idea of Kahneman's studies, we introduce here an enlightening experiment carried on during his research. Here, a group of people were asked to solve two quite easy tasks: *(i)* individuating the emotion in a human face; and *(ii)* calculating the result of a non-trivial multiplication (i.e. 17×23). This is depicted in Figure 1.1. All the participants showed a pretty identical behaviour: they were able to answer immediately to the image recognition task, while they all showed evident sign of effort (i.e. pupil dilatation) in solving the multiplication, which took evidently more time to be solved. From a computational viewpoint, there are no doubts that the multiplication is a far easier task than a face recognition task. So, why do humans show this behaviour? Leaving apart physiology and hardware differences (which are clearly very important), the most interesting analysis is that these two reasoning processes are, in humans, very different in the way humans tackle them. The *angry-face* task is an associative tasks; we use visual features to index some memory we have about angry faces, without the need to ask for more complex analyses. This "hashing" mechanism is demanding very low effort, since no complex or sequential algorithm should be run. On the other side, while we do



$$17 \times 23$$

Figure 1.1: Illustration of an experiment underlining the difference between symbolic and subsymbolic task. The recognition of the image emotion, which is considered a computationally harder task, is solved much faster by humans than a non-trivial multiplication. This is due to the fact that in solving the image recognition task we are making a lot of use of fast-associative inference, which is not possible in a purely symbolic multiplication task.

have very fast memories about 1 digit $\times$ 1 digit multiplication, we cannot store all the results of all the possible multiplications[1], so we need to apply a more complex algorithm, which allows us to generalize the simple multiplications we know to general and unseen cases. However, this process requires multiple steps, where intermediate results need to be stored and eventually integrated, and this is far more energy demanding than the previous task. By using Kahneman's definition, System 1 is capable of providing very reliable intuitions in the face emotion recognition task, while it is completely inadequate for solving the multiplication task, which is carried on by System 2.

An high level description of System 1 and System 2 behaviours and interaction is quoted from Kahneman (2011):

> System 1 operates automatically and quickly, with little or no effort and no sense of voluntary control. System 2 allocates attention to the effortful mental activities that demand it, including complex computations. The operations of System 2 are often associated with the subjective experience of agency, choice, and concentration. The automatic operations of System 1 generate surprisingly complex patterns of ideas, but only the slower System 2 can construct thoughts in an orderly series of steps. [...] Systems 1 and 2 are both active whenever we are awake. System 1 runs automatically and System 2 is normally in a comfortable low-effort mode,

[1] While this is a good and general rule of thumb, it is clearly not always true and we link here to the following subsection, about tight integration between symbolic and subsymbolic reasoning. Indeed, a shop assistant, who has to compute receipts of carts with lots of pieces all with a unary cost of 23, is likely to answer very fast to the multiplication above, as fast as we can answer 2×3. This is due to the fact that the shop assistant switches the paradigm in solving this task and their reasoning system figures out that it is convenient to store all the results if they are needed very frequently.

> in which only a fraction of its capacity is engaged. System 1 continuously generates suggestions for System 2: impressions, intuitions, intentions, and feelings. If endorsed by System 2, impressions and intuitions turn into beliefs, and impulses turn into voluntary actions. When all goes smoothly, which is most of the time, System 2 adopts the suggestions of System 1 with little or no modification. You generally believe your impressions and act on your desires, and that is fine—usually. When System 1 runs into difficulty, it calls on System 2 to support more detailed and specific processing that may solve the problem of the moment. System 2 is mobilized when a question arises for which System 1 does not offer an answer. You can also feel a surge of conscious attention whenever you are surprised. System 2 is activated when an event is detected that violates the model of the world that System 1 maintains.

It is extremely curious that lots of the tasks associated with System 1 in Kahneman (2011) are indeed tasks that deep learning techniques solve increasingly better as time goes by. And it is even more curious that the tasks associated with System 2 are instead tasks of control and complex reasoning, tasks that characterized the first period of AI and that failed when compared with the great success of deep learning. An interpretation of this failure in terms of Kahneman's studies is that complex reasoning in humans is not an abstract and isolated process but it is likely to be a second step of a simpler reasoning process based on intuitions, associations, approximations. Several Kahneman's experiments showed that human behaviour is mostly controlled by System 1, but that at the same time, System 1 needs System 2 in real complex scenarios, where multiple intuitions need to be ordered in a structured thought.

Coming back to artificial intelligence, we have two big subfield of artificial intelligence. From one side, we have what is currently referred to as *symbolic reasoning*. It is the field collecting techniques that exploit complex reasoning over abstract symbols to solve problems. It characterized the first era of AI and it is usually strictly linked with logical reasoning and deductive processes. The main problem of these techniques is their complexity, making them not usable in real world scenarios. On the other side we have the subfield of AI which is often referred to *subsymbolic* reasoning, which usually collects methods of solving tasks by exploiting (multiple) perceptual experiences to come up with a solution to a task. It is strictly connected with the fields of machine learning and data mining, and it has been recently pushed extremely forward by the techniques of deep learning. It is extremely more scalable than the first one, but at the same time lacks from abstraction power, making it extremely data hungry and inappropriate in complex scenarios.

The conceptual correlation between subsymbolic reasoning and System 1 and symbolic reasoning and System 2 should now be evident. And equally evident

should be our belief that, in designing a theory of artificial intelligence, both the field of symbolic reasoning and subsymbolic reasoning should have a role. Subsymbolic reasoning should guide the most of the reasoning processes, which allow the artificial agent to be able to take fast and effortless decisions. At the same time, symbolic reasoning should be responsible of checking whether the decisions of the subsymbolic component are feasible and refining them when necessary.

A tight integration. Another interesting notion that we can borrow from behavioural sciences concerns the fact that tasks accomplished by System 1 in one individual are not exactly the same tasks accomplished by System 1 in another individual. They coincide in most of the cases but some exceptions are possible depending on the individuals. This is extremely interesting since it seems suggesting that people can tackle the same tasks in different ways, some exploiting more intuition and memory association, others exploiting more control and logical reasoning. The most common example is that of people playing chess and usually showing signs of effort when thinking about a move (e.g. pupils dilatation); on the other side, chess champions usually play in a much more intuitive way, not always perfectly conscious about the reasons behind a certain move.

This circumstance seems to suggest that the surface of separation between which tasks are symbolic and which sub-symbolic is not perfectly delineated but reveals very blurred contours. And this behaviour is extremely delightful, in that there should be a unified theory behind both worlds. Abstract reasoning is likely to be the starting point in facing a new problem, due to the fact that abstraction allows generalization to never-seen tasks. However repetition and exercise slowly decrease the effort in solving the task and increase the intuitions about it. In Kahneman's words, System 1 needs always less and less supervisions from the System 2 and it becomes more and more proficient in a certain task, due to exercise and experience.

These considerations suggest a second desired feature of the theory we are going to provide. There should be a *unified principle* which allows a smooth transition between symbolic and subsymbolic reasoning.

1.2 State-of-the-art and Open Problems

We will go through several state-of-the-art systems in Chapter 2. Here we will give a general overview of where we are now in merging symbolic and subsymbolic reasoning and what problems are still open.

The task of merging symbolic and subsymbolic reasoning has a long history but it has taken a new look after the deep learning explosion as a new AI research subfield, named neural-symbolic integration (Garcez et al., 2012). There are two major ways of performing this integration.

- From one side, we want to *exploit neural techniques to improve purely symbolic tasks*. There are multiple ways of achieving this improvement. First of all, neural models are usually much faster in performing inference, thanks to their particular layered-structure of hidden variables; therefore, they can be effectively used to perform a fast approximate inference (Chang and Blei, 2009; Choi and Amir, 2012; Tu and Gimpel, 2018). Second, neural models usually deal with perceptual (i.e. subsymbolic) representations of elements of the world under investigation. This allows to exploit the particular geometry of the perceptual world to simplify inference (Diligenti et al., 2017). Indeed, at the end of the day, except very isolated cases, humans themselves reason about the world around them and not about abstract entities. Finally, when this perceptual space is not known, lot of neural models still assumed its existence and they attributed a vectorized representation to symbolic entities , which is optimized as a parameter of the learning problem. These representations are often referred to as embeddings and multiple approaches translate relational (i.e. logical) operations in terms of embeddings operations (Bordes et al., 2013; Sourek et al., 2015; Trouillon et al., 2016; Rocktäschel and Riedel, 2017).

- From the other side, we want to *exploit symbolic techniques to relate multiple neural tasks*. Indeed, in the setting of multi-task learning, there exists multiple examples of neural models solving multiple tasks simultaneously (Collobert et al., 2011). However, in most of (if not all) the cases tasks are not isolated but are related to each other. This is sometimes referred to as *structured prediction* or *structured learning*. The structure, even though adding a layer of complexity, can strongly improve neural models since structure is a compact way to describe lot of data (Liang et al., 2008). Indeed, one of the major drawback of neural models, and deep learning in general, is the extremely large amount of data they require to be trained. Structure is an high level source of knowledge, much more clean and valuable than single examples. Incorporating structure in deep learning is a fundamental step in improving its performances (Huang et al., 2015). Logic has always been considered one of the most elegant languages for expressing complexly structured knowledge and there have been multiple approaches trying to incorporate logic knowledge in deep learning models (Diligenti et al., 2017; Manhaeve et al., 2018).

Even though the awareness of the need for integrating models from the symbolic AI community and the deep learning community has been increasing a lot recently, the effort in the development of a unifying theory is still missing. While every single contribution is extremely valuable, lots of them are very uncorrelated, difficult to compare in order to underline common features or differences. This is likely to be due to the wide heterogeneity of skills of researchers in the area. This circumstance

makes the need of a unified and broad theory that describes multiple approaches and that can help understanding the connections between them.

Moreover, a sound integration solution should recover a symbolic approach in purely symbolic tasks and a subsymbolic approach in purely subsymbolic tasks, while most of the existent approaches are still very close to only one of the two classes of tasks.

Finally, the integration between the two layers is often achieved (e.g. in Bach et al. (2017)) by simply stacking a symbolic approach on top of a subsymbolic approach. This will hardly recover the tight integration that has been discussed in the previous Section, where we expect both layers to benefit from each other.

1.3 Contributions

The major contributions of the thesis are as follows:

- The introduction of the principle of MiniMax Entropy, where the maximization of the entropy for an unbiased description of the observations of the world is coupled with a local minimization of the same entropy aiming at selecting the best features for the description.
- Extension of the MiniMax Entropy principle to the conditional case, where perceptual data are associated to basic elements of the reasoning process. This extension individuates MiniMax Entropy models as a candidate for a unified theory of neuro-symbolic integration.
- Proposal of a new theory for the design of logical potential functions based on the exploitation of generated t-norm functions for the translation of FOL formulas into real-valued constraints, enabling differential inference methods. *Based on Giannini et al. (2019); Marra et al. (2019c)*.
- Characterization of Learning From Constraints models (Gnecco et al., 2015; Gori, 2017) as a very specific instance of MiniMax Entropy models in a variational inference setup.
- Definition, Implementation and Evaluation of LYRICS, a new programming language for describing a wide range of neuro-symbolic tasks using First Order Logic as user-interface language. *Based on Marra et al. (2019b)*.
- Proposal and evaluation of Deep Logic Models, MiniMax Entropy models exploiting fuzzy potentials and MAP inference. *Based on Marra et al. (2019a)*.
- Proposal and evaluation of Neural Markov Logic Networks, a MiniMax Entropy model with trainable relational potentials and Gibbs Sampling Inference. *Based on Marra and Kuželka (2019)*.

1.4 Structure of the thesis

This thesis is structured as follows.

- Chapter 2 introduces basic concepts necessary for the rest of the dissertation. In particular, two standard relational setting are described, namely First Order Logic and Logic Programming. Then, basic approaches of Statistical Relational Learning are described. They represent the background of many of the neuro-symbolic integration approaches described in the next part of the Chapter. Finally, other related approaches are discussed.

- Chapter 3 introduces the relational setting which will be shared by all the approaches of the thesis. Then, the basic principles of Maximization of Entropy and Minimization of Entropy are discussed and their integration into MiniMax Entropy problems is introduced, together with a general scheme for estimation and computation of these problems. The conditional MiniMax Entropy problem is discussed next, highlighting how this formulation makes MiniMax Entropy models a candidate for general models in neuro-symbolic integration. Finally, some approximated inference methods, exploited in the rest of the thesis, are described and some links to similar methods are underlined.

- Chapter 4 describes how fuzzy logics can be exploited to define potentials based on First Order Logic languages. In particular, the theory of t-norm and their generators is introduced and its declination in binding logic and learning is described.

- Chapter 5 introduces LYRICS, a general interface layer between logic and learning. It exploits the fuzzy logic conversion previously described to set up actual learning problems. The programming framework of LYRICS is described next. Finally, a large number of learning problem implemented in LYRICS are described, showing both the easiness of implementation and the competitiveness w.r.t. to other methods in the literature.

- Chapter 6 introduces Deep Logic Models (DLM), which are conditional MiniMax Entropy models exploiting fuzzy potentials and MAP inference as main ingredients. DLM are compared with state-of-the-art neuro-symbolic integration approaches, showing improvements both in structured learning tasks and link predictions tasks.

- Chapter 7 introduces Neural Markov Logic Networks (NMLN), which are MiniMax Entropy models exploiting parametric potentials and Gibbs Sampling as inference method. NMLN are allowed to learn the structure of the problem as neural potentials, allowing a very fine-grained representation of the underlying

distribution. NMLN are shown to compare favourably w.r.t. state-of-the-art neural theorem provers in link prediction tasks and to allow facing generative tasks in the relational setting.

- Chapter 8 draws the conclusions of the presented work and introduces possible future improvements of MiniMax Entropy problems.

Chapter 2

Related Works

2.1 Background

2.1.1 First Order Logic

In First Order Logic, a *term* is a variable, a constant or a functor applied to terms. An *atom* is of the form $p(t_1, \ldots, t_n)$ where p is a predicate of arity n and the t_i are terms. A *formula* is built out of atoms using universal and existential quantifiers and the usual logical connectives $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$. A *FOL theory* is a set of formulas that implicitly for a conjunction. An expression is called *ground* if it does not contain variables. A *literal* is an atom or its negation. Each disjunction of literals is said to be a *clause*. A disjunction consisting of a single literal is called a *unit clause*.

The *Herbrand base* of a FOL theory is the set of all ground atoms constructed using the predicates, functors and constants of the theory. A *Herbrand interpretation*, also called a *(possible) world*, is an assignment of a truth value to all atoms in the Herbrand base. A world or interpretation is called a *model* of the theory if it satisfies all formulas in the theory (in other worlds, if all formulas evaluate to true in that world).

2.1.2 Logic Programming (LP)

Syntactically, a normal logic program, or briefly *logic program (LP)* is a set of rules. A *rule* (also called a *normal clause*) is a universally quantified expression of the form `h :- b1, ... , bn`, where h is an atom and $b_1, \ldots, b_n$ are literals. The atom h is called the *head* of the rule and $b_1, \ldots, b_n$ the *body*, representing the conjunction $b_1 \wedge \ldots \wedge b_n$. A *fact* is a rule that has *true* as its body and is written more compactly as `h`.

When using the *well-founded semantics* for LPs (**?**) and in the case of a negation-free LP (or *definite program*), the well-founded model is identical to the well-known *Least Herbrand Model (LHM)*. The LHM is equal to the least of all models obtained when interpreting the LP as a FOL theory of implications. The *least* model is the model that is a subset of all other models (in the sense that it makes the fewest atoms

true). Intuitively, the LHM is the set of all ground atoms that are entailed by the LP. For negation-free LPs, the LHM is guaranteed to exist and be unique. For LPs with negation, the well-founded model (Van Gelder et al., 1991) is commonly used.

Intuitively, the reason why one considers only the *least* model of an LP is that LP semantics makes the *closed world assumption (CWA)*. Under the CWA, everything that is not implied to be true is assumed to be false. This has implications on how to interpret rules. Given a ground LP and an atom a, the set of all rules with a in the head should be read as the *definition* of a: the atom a is defined to be true if and only if at least one of the rule bodies is true (the 'only if' is due to the CWA). This means that there is a crucial difference in semantics between LP and FOL since FOL does not make the CWA. For example, the FOL theory $\{a \leftarrow b\}$ has 3 models $\{\neg a, \neg b\}$, $\{a, \neg b\}$ and $\{a, b\}$. The LP $\{$`a :- b`$\}$ has only one model, namely the least Herbrand model $\{\neg a, \neg b\}$ (intuitively, a and b are false because there is no rule that makes b true, and hence there is no applicable rule that makes a true either).

2.2 Related Statistical Relational Learning Approaches

2.2.1 MLN

Markov logic networks (MLNs) (Richardson and Domingos, 2006) 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. MLNs can be exploited to mostly carry out both inference and weight learning of the logical rules involved in a learning process, however in (Kok and Domingos, 2005) an algorithm to learn also the set of rules from scratch if presented, as well. 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 (Gutiérrez-Basulto et al., 2018), where MLNs are extended to deal with statistical universal quantifiers.

In particular, MLNs rely on the notion of Markov random field. 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. For each variable it is considered 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 2.1 (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 potential 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. 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 the lowest probability if it violates all the formulas. 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 2.2 (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 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 $\boldsymbol{x}$.

At hand, 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 schemes for MAP inference. That is a reason why

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

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 $j = 1, \ldots, m$, $\phi_j(\boldsymbol{x})$ 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 $I_j^+, I_j^- \subseteq \{1, \ldots, n\}$ be te set of indexes of the variables x_i occurring in C_j. Then C_j can be written as:

$$\left(\bigvee_{i \in I_j^+} x_i \right) \bigvee \left(\bigvee_{i \in I_j^-} \neg x_i \right) . \tag{2.1}$$

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:

$$\arg \max_{x \in \{0,1\}^n} \sum_{C_j \in \boldsymbol{C}} \omega_J \min \left\{ \sum_{i \in I_j^+} x_i + \sum_{i \in I_j^-} (1 - x_i), 1 \right\} . \tag{2.2}$$

While this program is generally intractable, some possible convex programming relaxations have been considered in the literature, e.g. (Bach et al., 2013, 2017). 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.

2.2.2 PSL

A possible way to make the problem in equation (2.2) 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 4.1) 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 (2.2) can be reformulated as follows.

$$\arg \max_{x \in [0,1]^n} \sum_{C_j \in \boldsymbol{C}} \omega_j \min \left\{ \sum_{i \in I_j^+} x_i + \sum_{i \in I_j^-} (1 - x_i), 1 \right\} . \tag{2.3}$$

Whenever a clause C_j is unsatisfied we can express its distance to satisfaction. In particular the problem in equation (2.3) can be rewritten in the following convex optimization problem:

$$\arg \min_{x \in [0,1]^n} \sum_{C_j \in \boldsymbol{C}} \omega_j \max \left\{ 1 - \sum_{i \in I_j^+} x_i - \sum_{i \in I_j^-} (1 - x_i), 0 \right\} , \tag{2.4}$$

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) (Bach et al., 2013, 2017), 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 (2.4) can be generalized to

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

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 2.3. *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*

$$\phi_j(\boldsymbol{y}, \underline{x}) = (\max\{l_j(\boldsymbol{y}, \boldsymbol{x}), 0\})^{p_j} \, , \tag{2.5}$$

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 2.4 (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:*

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

where

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

and $\boldsymbol{f}_{\boldsymbol{\omega}}$ *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 2.5. *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.*

Some comments are in orderd. 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 (2.5) 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 (Bach et al., 2017).

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:

$$\begin{aligned} \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \equiv \arg\min_{\boldsymbol{y}} \boldsymbol{f}_{\omega}(\boldsymbol{y},\boldsymbol{x}) = \arg\min_{\boldsymbol{y}} \boldsymbol{\omega}^T \cdot \boldsymbol{\phi}(\boldsymbol{y},\boldsymbol{x}) \\ \text{such that:} \qquad c_k(\boldsymbol{y},\boldsymbol{x}) \leq 0,\ \forall k \in \{1,\ldots,m\} \end{aligned} \tag{2.6}$$

MAP is a very fundamental task for PSL, indeed it allows the method to make predictions. In addition, as expressed by equation (2.6) 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

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

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 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.

2.2.3 ProbLog

ProbLog is a probabilistic extension of Prolog. ProbLog is essentially Prolog where all clauses are labeled with the probability that they are true, and these probabilities are

mutually independent. A ProbLog program specifies a probability distribution over all possible non-probabilistic subprograms of the ProbLog program. The semantics of ProbLog is then defined by the success probability of a query, which corresponds to the probability that the query succeeds in a randomly sampled program from the distribution.

ProbLog syntax

A ProbLog program consists of two parts: a set of ground probabilistic facts, and a logic program, i.e. a set of rules and ('non-probabilistic') facts. A ground *probabilistic fact*, written `p::f`, is a ground fact `f` annotated with a probability `p`. Syntactic sugar for compactly specifying an entire set of probabilistic facts with a single statement is usually allowed. Concretely, *intensional* probabilistic facts are allowed, which are statements of the form `p::f(X1,X2,...,Xn) :- body`, with `body` a conjunction of calls to non-probabilistic facts. The idea is that such a statement defines the domains of the variables `X1, X2, ...` and `Xn`. When defining the semantics, as well as when performing inference or learning, an intensional probabilistic fact should be replaced by its corresponding set of ground probabilistic facts, as illustrated below. An atom that unifies with a ground probabilistic fact is called a *probabilistic atom*, while an atom that unifies with the head of some rule in the logic program is called a *derived atom*.

The program for defining the classical `alarm` Bayesian network is given.

```
0.1::burglary.                          person(mary).
0.2::earthquake.                        person(john).
0.7::hears_alarm(X) :- person(X).
alarm :- burglary.
alarm :- earthquake.
calls(X) :- alarm, hears_alarm(X).
```

ProbLog Semantics

A ProbLog program specifies a probability distribution over possible worlds. To define this distribution, it is easiest to consider the grounding of the program with respect to the Herbrand base. Beforehand, a preprocessing step already replaced the intensional probabilistic facts with their corresponding ground set.

Each ground probabilistic fact `p::f` gives an *atomic choice*, i.e. one can choose to include f as a fact (with probability p) or discard it (with probability $1 - p$). A *total choice* is obtained by making an atomic choice for each ground probabilistic fact. Formally, a total choice is any subset of the set of all ground probabilistic atoms. Hence, if there are n ground probabilistic atoms then there are 2^n total choices. Moreover, we have a probability distribution over these total choices: the probability

of a total choice is defined to be the product of the probabilities of the atomic choices that it is composed of (the product can be taken since atomic choices are seen as independent events).

Given a particular total choice C, we obtain a logic program $C \cup R$, where R denotes the rules in the ProbLog program. We denote the well-founded model of this logic program as $WFM(C \cup R)$.[2] We call a given world ω a *model* of the ProbLog program if there indeed exists a total choice C such that $WFM(C \cup R) = \omega$.

Finally, the probability of a world that is a model of the ProbLog program is equal to the probability of its total choice; the probability of a world that is not a model is 0.

Inference tasks in ProbLog

In the literature on probabilistic graphical models and statistical relational learning, the two most common inference tasks are computing the marginal probability of a set of random variables given some observations or evidence (we call this the MARG task), and finding the most likely joint state of the random variables given the evidence (known as the MPE task, for Most Probable Explanation). In PLP, the focus has been on the special case of MARG where there is only a single query atom Q and no evidence.

All these tasks can tackled in ProbLog, with the addition of a new task, called EVID, that can be considered a building block for MARG. We now formally define these tasks. Let $\mathbf{At}$ be the Herbrand base, i.e, the set of all ground (probabilistic and derived) atoms in a given ProbLog program. We assume that we are given a set $\mathbf{E} \subset \mathbf{At}$ of observed atoms and a vector $\mathbf{e}$ with their observed truth values. We refer to this as the *evidence* and write $\mathbf{E} = \mathbf{e}$. Note that the evidence is essentially a partial interpretation of the atoms in the ProbLog program.

- In the **MARG** task, we are given a set $\mathbf{Q} \subset \mathbf{At}$ of atoms of interest, called *query atoms*. The task is to compute the marginal probability distribution of every such atom given the evidence, i.e. compute $P(Q \mid \mathbf{E} = \mathbf{e})$ for each $Q \in \mathbf{Q}$.

- The **EVID** or 'probability of evidence' task is to compute $P(\mathbf{E} = \mathbf{e})$. It corresponds to the likelihood of data in a learning setting and can be used as a building block for solving the MARG task.

- The **MPE** task is to find the most likely interpretation (joint state) of all non-evidence atoms given the evidence, i.e. finding $\arg\max_{\mathbf{u}} P(\mathbf{U} = \mathbf{u} \mid \mathbf{E} = \mathbf{e})$, with $\mathbf{U}$ being the unobserved atoms, i.e., $\mathbf{U} = \mathbf{At} \setminus \mathbf{E}$.

The ProbLog approach to inference consists of two steps:

[2]Recall from Section 2.1.2 that for negation-free programs, the WFM is the least Herbrand model.

1. **Conversion of the program to a weighted Boolean formula**. The conversion takes as input a ProbLog program L, evidence $\mathbf{E} = \mathbf{e}$ and a set of *query atoms* $\mathbf{Q}$, and returns a weighted Boolean (propositional) formula that contains all necessary information. The conversion is similar for each of the considered tasks (MARG, MPE or EVID). The only difference is the choice of the query set $\mathbf{Q}$. For MARG, $\mathbf{Q}$ is the set of atoms for which we want to compute marginal probabilities. For EVID and MPE, $\mathbf{Q} = \emptyset$. The outline of the conversion algorithm is as follows.

   a) *Ground L yielding a program L_g while taking into account $\mathbf{Q}$ and $\mathbf{E} = \mathbf{e}$.*
   It is unnecessary to consider the full grounding of the program, we only need the part that is relevant to the query given the evidence, that is, the part that captures the distribution $P(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$. We refer to the resulting program L_g as the *relevant ground program* with respect to $\mathbf{Q}$ and $\mathbf{E} = \mathbf{e}$.

   b) *Convert the ground rules in L_g to an equivalent Boolean formula φ_r.*
   This step converts the logic programming rules to an equivalent formula and it is a purely logical processing.

   c) *Assert the evidence and define a weight function*
   To obtain the weighted formula, we first assert the evidence by defining the formula φ as the conjunction of the formula φ_r for the rules and for the evidence φ_e. Then we define a *weight function* for all atoms in φ, where the weight is strictly connected to probabilities.

2. **Inference on the weighted Boolean formula.** At this stage, there is no more traces of the logic program. We are left with a formula and a weighting function. Thus the different inferences tasks are reformulated in terms of well-known tasks such as MAX-SAT (for MPE) or weighted model counting (for EVID and MARG)

As a final note, for solving weighted model counting, ProbLog makes use of an intermediated step which translate ("compile") the formula into a more efficient one, i.e. an arithmetic circuit, which is closely linked to the concept of deterministic, decomposable negation normal form (d-DNNF).

ProbLog and MLNs

There are some very interesting connections between a ProbLog program and a specially constructed MLN. Indeed, the weighted formula that ProbLog construct can be regarded as a ground MLN. The MLN contains the Boolean formula as a "hard" formula (with infinite weight). The MLN also has two weighted unit clauses per probabilistic atom: for a probabilistic atom a and weight function $\{a \mapsto p, \neg a \mapsto$

$1 - p\}$, the MLN contains a unit clause a with weight $\ln(p)$ and a unit clause $\neg a$ with weight $\ln(1 - p)$.

There is the following equivalence result.

Theorem 1. *Let L_g be the relevant ground program for some ProbLog program with respect to $\mathbf{Q}$ and $\mathbf{E} = \mathbf{e}$. Let $\mathcal{M}$ be the corresponding ground MLN. The distribution $P(\mathbf{Q})$ according to $\mathcal{M}$ is the same as the distribution $P(\mathbf{Q} \mid \mathbf{E} = \mathbf{e})$ according to L_g.*

Note that for the MLN the distribution $P(\mathbf{Q})$ (not conditioned on the evidence) is considered. This is because the evidence is already hard-coded in the MLN.

2.3 Neuro-Symbolic Integration

2.3.1 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) (Diligenti et al., 2017), 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 (Diligenti et al., 2017). 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.

Let us consider a set of predicate functions $\boldsymbol{P} = \{p_1, \ldots, p_J\}$, 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 4.1. The universal and existential quantifiers over a variable x_i are converted according to the following expressions:

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

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

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

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

Definition 2.6 (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. We denote by $\boldsymbol{x}$ the feature vector associated to a grounding x, with 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 $\boldsymbol{f}$ to belong to a certain functional space $\mathcal{H}_j$, 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:

$$C[\boldsymbol{f}(\mathcal{X})] = \sum_{j=1}^{J} ||f_j||^2_{\mathcal{H}_j} + \sum_{h=1}^{H} \lambda_h (1 - \Phi_h(\boldsymbol{f}(\mathcal{X}))) \; . \tag{2.7}$$

However, the cost function in equation (2.7) is in general not convex, indeed 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 Chapters 4 and 5 can be considered as an extension of SBR.

2.3.2 DeepProbLog

a-ProbLog Analyzing the probabilistic Prolog from an algebraic point of view reveals that the probabilities associated to facts and queries are essentially elements of $\mathbb{R}_{\geq 0}$ and the operations needed to compute the success probability of a query are addition and multiplication, which means that one is operating in the semiring $(\mathbb{R}_{\geq 0}, +, \times, 0, 1)$. This raises the question as to whether it is possible to generalize these probabilistic Prologs to use labels from different semirings. Kimmig et al. (2011) introduces aProblog, i.e. algebraic Problog, which generalizes the probabilistic programming language ProbLog. An aProbLog program consists of a set of definite clauses and a set of algebraic facts. These are facts that are labeled with elements of a commutative semiring $\mathcal{R}$. The label of a possible world is then simply the product (in $\mathcal{R}$) of the labels of the algebraic literals it contains. The label of a query is the sum (in $\mathcal{R}$) of the labels of the possible worlds in which the query succeeds.

One interesting labeling scheme is the the gradient of the success probability, which makes use of the gradient semiring. The gradient of the success probability is already used in parameter learning in ProbLog but aProblog provides a general view of its computation.

DeepProbLog Manhaeve et al. (2018) introduced DeepProbLog, an extension of ProbLog to allow the use of neural predicates. In DeepProbLog, the probabilities given to probabilistic atoms can also be predicted by neural networks, which operates on an input representation of the constant.

In contrast to the earlier approach for ProbLog parameter learning, DeepProbLog uses gradient descent rather than EM, as this allows for a seamless integration with neural network training. Given a LP program with neural predicates, its neural network models, and a query used as training example, DeepProbLog first grounds

the program with respect to the query, getting the current parameters of nADs from the external models, then uses the ProbLog machinery to compute the loss and its gradient, and finally uses these to update the parameters in the neural networks and the probabilistic program.

In particular, the use of neural predicates as an interface between the logic and the neural part allows both sides to treat the other as a black box. The logic side can calculate the gradient of the loss w.r.t. the output of the neural network, but is unaware of the internal parameters. However, the gradient w.r.t. the output is sufficient to start backpropagation, which calculates the gradient for the internal parameters. Then, standard gradient-based optimizers (e.g. SGD, Adam, ...) are used to update the parameters of the network.

Discussion DeepProbLog is without no doubt one of the most promising framework in the neuro-symbolic integration literature. However, in its current shape, the framework suffers for several drawbacks which need to be tackled. By using aProblog as basic recipe for probabilistic inference and gradient computation, DeepProbLog cannot scale to problems of the size of current deep learning benchmarks. According to the author of this thesis, this should never be a limit for a framework, but a scheme, which gracefully degrades performance (in terms of approximation) in favour of scalability should be taken into account. Another drawback of the system is related to the Closed World Assumption, which makes it not ideal as a general purpose AI framework. By moving to real world size, it is necessary to consider that something is simply unknown but that it can still be true. The integration of priors for unknown facts could be a very interesting evolution. Indeed, it has been shown that the possibility of tuning priors in MLNs allows it to outperform ProbLog in a link prediction task.

2.3.3 Lifted Relational Neural Networks

Sourek et al. (2015) propose a method combining relational-logic representations with neural network learning. A general lifted architecture, possibly reflecting some background domain knowledge, is described through relational rules which may be handcrafted or learned. The relational rule-set serves as a template for unfolding possibly deep neural networks whose structures also reflect the structures of given training or testing relational examples. Different networks corresponding to different examples share their weights, which co-evolve during training by stochastic gradient descent algorithm. The framework allows for hierarchical relational modeling constructs and learning of latent relational concepts through shared hidden layers weights corresponding to the rules.

A lifted relational neural network (LRNN) $\mathcal{N}$ is a set of weighted definite clauses, i.e. pairs (R_i, w_i) where R_i is a function-free definite clause and w_i is a real number.

When $\mathcal{N}$ is a set of weighted definite clauses, $\mathcal{N}^*$ will denote the corresponding set of the definite clauses without weights, i.e. $\mathcal{N}^* = \{C : (C, w) \in \mathcal{N}\}$. The set $\mathcal{N}$ must satisfy the following *non-recursiveness* requirement: there must exist a strict ordering $\prec$ of predicates such that if there is a rule with a predicate p_1 in the head and a predicate p_2 in the body then $p_1 \prec p_2$.

Given a LRNN $\mathcal{N}$, let $\mathcal{H}$ be the least Herbrand model of $\mathcal{N}^*$. We define *grounding of the LRNN* $\mathcal{N}$ as $\overline{\mathcal{N}} = \{(h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta, w) : (h \leftarrow b_1 \wedge \cdots \wedge b_k, w) \in \mathcal{N} \text{ and } \{h\theta, b_1\theta, \ldots, b_k\theta\} \subseteq \mathcal{H}\}$. That is, $\overline{\mathcal{N}}$ is defined as the set of ground definite clauses which can be obtained by grounding rules from the LRNN and which are active in the least Herbrand model of $\mathcal{N}^*$ (a rule is active in $\mathcal{H}$ if its body is true in $\mathcal{H}$). LRNNs are templates for creating *ground* neural networks. The requirement that ground rules should be active in $\mathcal{H}$ is beneficial for practice because it provides us with flexibility in controlling complexity of the constructed neural networks.

Definition 2.7. *Let $\mathcal{N}$ be a LRNN, and let $\overline{\mathcal{N}}$ be its grounding. Let $g_\vee$, $g_\wedge$ and $g^*_\wedge$ be families of multivariate functions with exactly one function for each number of arguments. The* ground neural network *of $\mathcal{N}$ is a feedforward neural network constructed as follows.*

- *For every ground atom h occurring in $\overline{\mathcal{N}}$, there is a neuron A_h, called* atom neuron. *The activation functions of atom neurons are from the family $g_\vee$.*
- *For every ground fact $(h, w) \in \overline{\mathcal{N}}$, there is a neuron $F_{(h,w)}$, called* fact neuron, *which has no input and always outputs a constant value.*
- *For every ground rule $h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta \in \overline{\mathcal{N}}^*$, there is a neuron $R_{h\theta \leftarrow b_1\theta \wedge \cdots \wedge b_k\theta}$, called* rule neuron. *It has the atom neurons $A_{b_1\theta}, \ldots, A_{b_k\theta}$ as inputs, all with weight* 1. *The activation functions of rule neurons are from the family $g_\wedge$.*
- *For every rule $(h \leftarrow b_1 \wedge \cdots \wedge b_k, w) \in \mathcal{N}$ and every $h\theta \in \mathcal{H}$, there is a neuron $Agg^{h\theta}_{(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$, called* aggregation neuron. *Its inputs are all rule neurons $R_{h\theta' \leftarrow b_1\theta' \wedge \cdots \wedge b_k\theta'}$ where $h\theta = h\theta'$ with all weights equal to* 1. *The activation functions of the aggregation neurons are from the family $g^*_\wedge$.*
- *Inputs of an atom neuron $A_{h\theta}$ are the aggregation neurons $Agg^{h\theta}_{(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)}$ and fact neurons $F_{(h\theta, w)}$. The weights of the input neurons are the respective w's.*

Depending on the used families of activation functions $g_\vee$, $g_\wedge$ and $g^*_\wedge$, one can obtain neural networks with different behavior. For intuitiveness, in order for rules $(h \leftarrow b_1 \wedge \cdots \wedge b_k, w)$ to behave similarly to "if-then" rules, we should prefer the outputs of rule neurons to be *high* (e.g. close to 1) if and only if all the inputs from the atom neurons corresponding to the literals from the body of the rule have high outputs. Similarly, we should prefer the output of the atom neurons, which should intuitively behave similarly to disjunction, to be high if and only if at least one of the

rule neurons or fact neurons, which are inputs for the given atom neuron, has high output. Logical operators from various fuzzy logics may serve as an inspiration for selecting suitable activation functions.

Two particular collections of activation functions inspired by fuzzy logic are introduced:

Definition 2.8 (Max-Sigmoid Activation Functions)**.** *The Max-Sigmoid (MS) collection of activation functions is composed of the following three families of functions:* $g_\wedge(b_1,\ldots,b_k) = sigm\left(\sum_{i=1}^{k} b_i - k + b_0\right)$, $g^*_\wedge(b_1,\ldots,b_m) = \max_i b_i$, *and* $g_\vee(b_1,\ldots,b_k) = sigm\left(\sum_{i=1}^{k} b_i + b_0\right)$.

The rationale for this family of activation functions is as follows. As already mentioned, the activation function $g_\wedge$ should have high output if and only if all its inputs are high. To achieve this, we can crudely approximate Lukasiewicz fuzzy conjunction, which is given as $\max\{0, b_1 + \cdots + b_k - k + 1\}$, by the function $sigm\,(b_1 + \cdots + b_k - k + b_0)$.

The activation function $g^*_\wedge$ outputs the value equal to the highest of its inputs. This can be seen as finding the best "match" of a pattern (rule).

The activation function $g_\vee$ should have high output if at least one of the inputs is high or if all inputs are somewhat high. To satisfy this, one can crudely approximate Lukasiewicz fuzzy disjunction, which is given as $\min\{1, b_1 + \cdots + b_k\}$ by the function $sigm\,(b_1 + \cdots + b_k + b_0)$.

The Max-Sigmoid activation function is obviously not the only one possible. It is useful when we are interested in detecting one or more patterns but less useful when we are more interested in a global view of all the patterns.

A family of activation functions which are more appropriate in these situations is given by the next definition.

Definition 2.9 (Avg-Sigmoid Activation Functions)**.** *The Avg-Sigmoid (AS) collection of activation functions is composed of the following three families of functions:* $g_\wedge(b_1,\ldots,b_k) = sigm\left(\sum_{i=1}^{k} b_i - k + b_0\right)$, $g^*_\wedge(b_1,\ldots,b_m) = \frac{1}{m}\sum_{i=1}^{m} b_i$, *and* $g_\vee(b_1,\ldots,b_k) = \sum_{i=1}^{k} b_i + b_0$.

Another advantage of the Avg-Sigmoid family of activation functions over the Max-Sigmoid family is also that the functions from the Avg-Sigmoid family are everywhere differentiable (which simplifies learning). We note that other activation function families based on combinations of different aggregation functions might also be exploited for LRNN learning.

Discussion Lifted Relation Neural Networks proposed a very interesting and original way of integrating standard logical reasoning with neural components. However, even though it is still possible, this model does not take into account the fact that constants can have a perceptual description and that exploiting this description can

strongly increase the reasoning capabilities of an intelligent agent. Moreover, the *per-example* construction of neural networks does not allow the system to exploit data-parallelism, which is a requirement of current GPU-computations.

2.3.4 Neural Theorem Provers

A very similar approach to LRNNs is the one proposed in Rocktäschel and Riedel (2017), where they propose a new neural network model for end-to-end differentiable proving of queries to knowledge base by operating on dense vector representation of symbols.

The basic idea, derived for other works in the deep learning community like Neural Turing Machines (Graves et al., 2014) or Memory Networks (Weston et al., 2014), is that of replacing discrete algorithms and data structures by end-to-end differentiable counterparts that operate on real valued vectors. In particular, they applied this approach to basic symbolic theorem provers (in particular, Prolog), thus combining their advantages (multi-hop reasoning, interpretability, easy integration of domain knowledge) with the ability to reason with vector representations of predicates and constants. They keep variable binding symbolic (discrete) but compare symbols using their subsymbolic vector representations.

Concretely, they introduced Neural Theorem Provers (NTPs) which are end-to-end differentiable provers for basic theorems formulated as queries to a KB. Prolog's backward chaining algorithm is used as the basic recipe for recursively constructing a neural network that is capable of proving a query to a KB using subsymbolic representations. The success score of this proof is differentiable with respect to vector representations of symbols, which enables the model to learn such representations for predicates and constants in ground atoms, as well as parameters of function-free first-order logic rules of predefined structure. By doing so, NTPs learn to place representations of similar symbols in close proximity in a vector space and to induce rules given prior assumptions about the structure (i.e. template) of logical relationships in a KB such as transitivity. Furthermore, NTPs can seamlessly reason with provided domain-specific rules. As NTPs operate on distributed representations of symbols, a single hand-crafted rule can be leveraged for many proofs of queries with symbols that have a similar representation. Finally, NTPs demonstrate a high degree of interpretability as they induce latent rules that we can decode to human-readable symbolic rules.

Discussion. Being very similar in spirit to LRNNs, NTPs suffer from similar problems. Another drawback of the system is concerned with the fact that, for answering a given query, this model needs to consider all possible proof paths, and then aggregate results. This quickly becomes infeasible even for small Knowledge Bases (KBs). A recent contribution (Minervini et al., 2018) shows that one can accurately

approximate the inference process in this model by considering only proof paths associated with the highest proof scores, enabling inference and learning on previously impracticable KBs.

2.4 Other related approaches

ILP. One of the best known research area that combines logic programming with machine learning techniques is *Inductive Logic Programming* (ILP) (Muggleton, 1991; Muggleton and De Raedt, 1994). 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 (Muggleton, 1994) assumes a prior probability distribution defined over the hypothesis space. In (De Raedt and Kersting, 2008) 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 (Landwehr et al., 2010) and (Muggleton et al., 2005). 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. In the second one a kernel, that is an inner product in the feature space spanned by a given set of first-order hypothesized clauses, is proposed.

ProPPR An interesting extension of LP that enables efficient learning and inference on graphs is given by ProPPR (Wang et al., 2013). 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 (Wang and Cohen, 2016), 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.

TensorLog In (Cohen, 2016) is presented a probabilistic deductive database, called TensorLog, in which reasoning uses a differentiable process. TensorLog is shown to be faster 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.

Neural LP An extension to jointly learn the structure of the rules is proposed by *Neural Logic Programming* (Neural LP) (Rocktäschel and Riedel, 2016) 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.

Integrating MLN and neural networks Lippi and Frasconi (2009) was an early attempt to integrate MLNs with neural components. Here, an MLN was exploited to describe a conditional distribution over ground atoms, given some features of the constants. In particular, the MLN parameters were predicted by a neural network evaluated on input features.

GNNs: extracting regularities in non-euclidean settings The idea of exploiting neural networks to extract regularities in non-euclidean settings has been recently revisited by the deep learning community in the context of Graph Neural Networks (GNN) models (Scarselli et al., 2009; Defferrard et al., 2016; Xu et al., 2018). In GNNs, latent representations of nodes are obtained by an aggregation of neighboring nodes representation by means of an iterative diffusion mechanism. However, the inference is performed only on neighborhoods induced by the actual connections of the graph, preventing the exploitation of these models for modeling distributions of structural properties of the graph.

Generative Energy-Based Models Generative Energy-Based models have been an AI hot topic since the early days of the connectionism. Classical approaches are Ackley et al. (1985); Salakhutdinov et al. (2007); Salakhutdinov and Hinton (2009). The main drawback of these models is that, in generative tasks, commonly exploited approximations tend to fail and thus their application is feasible only in restricted domains. However, these models are gaining attention again nowdays (Du and Mordatch, 2019) and they are starting to be applied to challenging generation tasks (e.g. images).

MaxEnt modeling in the relational setting One important piece of the proposed approach is the ability of modeling the feature binding problem of an exponential distribution using the MaxEnt principle. This principle has been largely investigated in the propositional setting, but few has been done in the relational setting. An interesting recent analysis is the one in Kuželka et al. (2018), where MaxEnt distributions in the relational setting are introduced. Here, authors showed how MLNs can be analyzed as those distribution with maximum entropy subject to relational marginals

constraints. The approach proposed in this thesis, in this regard, can be considered an extension of Kuželka et al. (2018), where MinEnt allows for the feature selection problem to be solved. Another related approach is the one in Kern-Isberner et al. (2017), where MaxEnt is used as inspiration for modeling probabilistic conditionals. While the premises are indeed related to the proposed method, the authors do not exploit dual optimization techniques and they keep the problem in its primal formulation. This will arguably prevent the application of this method to large scale problems.

Chapter 3

A Mini-Max Entropy framework in the relational setting

In this chapter, we show a general theory to integrate symbolic and subsymbolic approaches, as motivated in Chapter 1.

The world we want to describe and reason about is described in a relational nature (Section 3.1) where elements of the domain of discourse are related to each other by some properties.

One of the main ingredient of the proposed approach is reasoning and discussing about the *truth value* of atomic elements of this relational structure, i.e. ground atoms (Section 3.1). Examples of these facts can be: `student(Giuseppe) = True`, `advisorOf(Marco,Giuseppe) = True`, `red(Sky) = False`. First Order Logic is a natural and expressive language to this end. A given truth assignment to all the facts we want to reason about is called a *possible world*.

We inherently admit uncertainty in our statements and in our reasoning tasks. We, then, assign probabilities to facts (Section 3.1). For example, $p($`blue(Sky) = True`$) = 0.99$ and $p($`red(Sky) = True`$) = 0.01$.

Structures guiding the reasoning process can be given or learned by complete or partial examples of possible worlds.

It follows that the main goal is modelling a probability distribution over truth assignments to all facts we can reason about, i.e. we want to model a probability distribution $p(\boldsymbol{y})$ over possible worlds $\boldsymbol{y}$.

This Chapter is structured as follows. Section 3.1 introduces basic concepts of a relational structure based on a FOL language, introducing the idea of factorization using subsets of the constants. Section 3.2 formally introduces the principle of MiniMax Entropy, both from an intuitive and formal viewpoint. On one side, the Maximization of Entropy drives the feature binding problem, where features of the relational structures are used as basic language to describe the provided data. On the other side, the Minimization of Entropy drives the feature selection problem,

where the most descriptive features are learned. In Section 3.3, we extend MiniMax Entropy models to the case in which constants of the FOL language are attached with a perceptual representation and it explores how we can exploit this representation to improve reasoning over the relational structure. In Section 3.4, we show how the theory of MiniMax Entropy nicely recovers standard learning settings in both supervised and statistical relational learning, which is considered a mandatory requirement of any theory aiming at merging symbolic and subsymbolic techniques (De Raedt et al., 2019). The Chapter is concluded with Section 3.3.1, where we describe a particular inference method in MiniMax Entropy models, i.e. MAP inference. It is interesting both from a theoretical point of view, since it links some very distant learning frameworks, from a computational viewpoint, since it implements a fast inference scheme, and a motivational viewpoint, since it provides a potential link to fast inference schemes in humans.

3.1 The relational setting

3.1.1 Constants and Predicates

We consider a function-free first-order logic language $\mathcal{L}$. It is built from a set of constants $\mathcal{C}$ and a set of predicates $\mathcal{R}$.

The *set of constant* $\mathcal{C}$ collects the inhabitants of the world we are describing. Let g be a function from $\mathcal{C}$ to $\mathbb{R}^m$. This is a function extracting a m-dimensional feature representation of constants. When coupling the identifier j of a constant c_j with its feature representation $x_j = g(c_j)$, we are modeling the fact that individuals of the world exist *per-se* (by giving them an abstract identifier) but they can also be described by some *profile*, i.e. by some collection of perceptual[1] measures representing the individual.

Example 3.1 (Constants)**.** *An example of constants can be the collection of people in a university. So,* $\mathcal{C} = \{\texttt{Giuseppe}, \texttt{Maria}, \texttt{Marco}\}$*. For each of these individuals, we could be interested in reasoning about some (perceptual) data, for example, their age, their weight, their height, the number of courses they passed, etc. All these pieces of information can be encoded into a real vector, e.g.* $g(\texttt{Giuseppe}) = [28, 76, 180, 25]$.

The *set of predicates* $\mathcal{R}$ can be defined as $\mathcal{R} = \bigcup_i \mathcal{R}_i$, where $\mathcal{R}_i$ contains the predicates of arity i. In our language, we can relate multiple constants by stating the validity of a certain property over them.

[1] With perceptual, we mean here coming from human-like senses or other measurements instruments. Examples of this kind of data are images, sounds, temperature index, brightness index, ecc. We use the perceptual word as a synonym of concrete.

Definition 3.1 (Ground Atom). *For $c_1, c_2, \ldots, c_m \in \mathcal{C}$ and $R \in \mathcal{R}_m$, we define ground atom the instantiation $R(c_1, c_2, \ldots, c_m)$.*

Example 3.2 (Predicates and Ground Atoms). *In our example of university people, we can introduce a set of properties we want to reason about. So let us define a set of predicates $\mathcal{R} = \{\texttt{student}(1), \texttt{professor}(1), \texttt{advise}(2), \texttt{sameAdvisor}(2)\}$, where we put in brackets their arity. By exploiting this set of relations we can state some facts about the world, which are technically called ground atoms. For example, in the language defined up to now, we can state that the following facts hold* `True: student(Giuseppe), student(Maria), professor(Marco), advise(Marco, Giuseppe)`*, where we are stating that* `Giuseppe` *and* `Maria` *are* `students`*,* `Marco` *is a* `professor` *and* `Marco` *is the advisor of* `Giuseppe`*. Usually when a fact is stated without a truth value, we are implicitly considering it as* `True`*.*

Given a language $\mathcal{L}$, and, thus, given a fixed set of constants and predicates, we call Herbrand Base the set of all ground atoms constructed using all constants and all the predicates in $\mathcal{L}$. A given assignment of truth values to the Herbrand base is called an Herbrand interpretation, commonly referred to as *possible world*.

We can use this language to describe many different worlds $\boldsymbol{y}$. For example, in a certain world $\boldsymbol{y}_1$, both `Giuseppe` and `Maria` are students, i.e. `student(Giuseppe), student(Maria)`. However, in another world, only `Giuseppe` is a student. In other words, given a language, we can instantiate many different possible worlds.

Definition 3.2 (Herbrand Base and Possible World). *We define the Herbrand Base of a FOL language composed of a set $\mathcal{C}$ of constants and a set $\mathcal{R}$ of relations the set $HB = \{R(c_1, \ldots, c_n) : (c_1, \ldots, c_n) \in \mathcal{C}^n \wedge R \in \mathcal{R}_n\}$, which is the set of all ground atoms definable in $\mathcal{L}$. A given truth assignment $\boldsymbol{y}$ to all the ground atoms in HB is said to be an Herbrand Interpretation or possible world.*

3.1.2 From structures to substructures

Suppose we are provided with a first-order language $\mathcal{L} = (\mathcal{C}, \mathcal{R})$ as described in the previous section. Suppose we are also provided with a specific world $\hat{\boldsymbol{y}}$. Remember our goal is the one of finding a model for the probability distribution $p(\boldsymbol{y})$ of a generic world. Since we are provided with only one example of this relational structure, i.e. $\hat{\boldsymbol{y}}$, a pure and naive frequentist approach is doomed to fail. How can we spot regularities or statistics of worlds if we are only provided by a single world? However, if we consider the nature as we know it, it is a single instantiation of many potential (possible) worlds, but we are still capable of extracting regularities which help us to reason about things in this particular world.

The main idea, which is shared by the majority of statistical relational learning approaches, to solve this dilemma is to consider our single world $\hat{\boldsymbol{y}}$ as composed by multiple substructures. The problem is then translated from spotting regularities at

the whole structure level to spotting regularities in the substructure level and exploit these regularities to model the probability distribution $p(\boldsymbol{y})$ of the entire structure. Thus, it naturally emerges the need to refer to substructures of the general world. We define these substructures as *fragments* of a possible world.

Definition 3.3 (Fragment)**.** *Let be $S \subset \mathcal{C}$. A fragment $\boldsymbol{y}\langle S\rangle$ is defined as the restriction of $\boldsymbol{y}$ to the constants in S. It can be considered a truth assignments to the subset of the restriction $HB\langle S\rangle$ of the HB, with $HB\langle S\rangle = \{R(c_1,\ldots,c_n), \forall n \leq |S| : \forall(c_1,\ldots,c_n) \in \mathcal{S}^n \wedge R \in \mathcal{R}_n\}$*

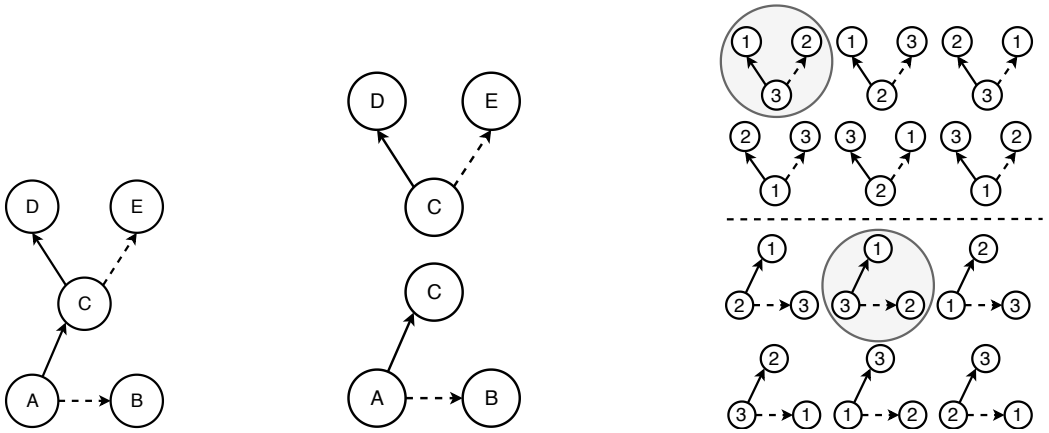
Fragments of a world are a first step towards enabling the possibility of spotting regularities of substructures of possible worlds. However, until constants keep their identity, their behaviour is still a singular one, denying any possibility of evaluating statistics. A fundamental step in this direction is the *anonymization* of fragments. When constants inside fragments are anonymized, only their relational structure survives. Thus, fragments with the same relational structure cannot be distinguished anymore, allowing us to compute statistics of the common relational patterns among constants.

Definition 3.4 (Anonymization of fragments)**.** *Given a fragment $\boldsymbol{y}\langle S\rangle$ and $k = |S|$, we define anonyzation the mapping of the constants in S to a permutation $\hat{S}$ of the integer set $\{1, 2, ..., k\}$. We define $\boldsymbol{y}\langle\hat{S}\rangle$ an anonymized fragment . Suppose we have a given world $\widehat{\boldsymbol{y}}$ of size n, we define $\Gamma_k(\widehat{\boldsymbol{y}})$ the collection of all the anonymized fragments of width k of $\widehat{\boldsymbol{y}}$.*

It is easy verifiable that $|\Gamma_k(\widehat{\boldsymbol{y}})| = \binom{n}{k}k!$. The collection $\Gamma_k(\widehat{\boldsymbol{y}})$ is a multiset, since, after anonymization, multiple fragments could be identical. It is worth noticing that it is exactly the multiplicity of this set to allow us to individuate some regularities among small fragments of the data, which can be exploited to model the data distribution. A representation of the process of anonymization and of the identification of structural patterns among anonymized fragments is shown in Figure 3.1. An example is provided instead in Example 3.3.

Example 3.3 (Fragments and Anonymization)**.** *Let define a set of constants $\mathcal{C} = \{$`Alice, Bob, Eve`$\}$ and a set of relations $\mathcal{R} = \{$`smokes`$(1),$ `friend`$(2)\}$. Consider the following possible world $\boldsymbol{y} = \{$`smokes(Alice), friend(Alice,Bob), friend(Bob,Eve)`$\}$. Given a fragment $\boldsymbol{y}\langle S\rangle$, with $S = \{$`Alice, Bob`$\}$, i.e. $\boldsymbol{y}\langle S\rangle = \{$`smokes(Alice), friend(Alice,Bob)`$\}$, an example of anonymized fragment of $\boldsymbol{y}\langle S\rangle$ is $\widehat{S} = \{1, 2\}$ and $\boldsymbol{y}\langle\widehat{S}\rangle =$ {`smokes(1), friend(1,2)`}, which can be paraphrased as " the first constant of the fragment smokes and it is a friend of the second constant".*

It is evident that if multiple fragments share the same relational structures they are not identifiable any more. Than, we can compute how many fragments share some relational pattern and use these statistics to model the probability distribution. The role of computing statistics is given to a class of functions, called *potential functions*.



(a) The global example $\boldsymbol{y}$

(b) Two $k = 3$ fragments.

(c) Anonymizations of the two fragments.

Figure 3.1: **The process of individuating structural patterns in anonymized fragments.** White circles represent constants, while the two relations are represented as solid and dashed arrows (absence of an arrow means that the relation is false). The given world is shown on the left. Two possible fragments are shown in the middle. All their possible anonymizations are shown on the right. *Grey circles* show two repeated anonymized fragments found in two different fragments. The model exploits these regularities on fragments to model the distribution of the larger structure.

Potential functions A fundamental ingredient, in our approach, as it will be evident shortly, is the capability of measuring *statistics* of properties (or features) of the worlds described by the language $\mathcal{L}$. Indeed, in order to design good models of the probability $p(\boldsymbol{y})$ we would like that samples from the probability distribution model show properties similar to the data we are observing.

The role of computing properties of worlds and fragments is given to a class of functions, called *potential* or *feature functions*.

We define two types of potential functions:

- the *global* or *world potential functions* Φ, which are functions from the set $\mathcal{Y}$ of the possible worlds to the real set $\mathbb{R}$;
- the *local* or *fragment potential functions* ϕ, which are functions from the set of fragments Γ_k to the real set $\mathbb{R}$.

Potential functions computes the value of some properties (or features) of worlds and fragments, respectively.

The definition of these potential functions is very general and this unconstraintness is fundamental since potentials functions represent the *alphabet* for describing our distribution. The finer their representativeness of the observed data, the best the model can be.

Example 3.4 (Potential Functions)**.** *Consider the set of constants* $C = \{\texttt{A},\texttt{B},\texttt{C},\texttt{D}\}$ *and two possible worlds* $\boldsymbol{y}_1$ *and* $\boldsymbol{y}_2$*:* $\boldsymbol{y}_1 = \{\texttt{friend(A,B), friend(B,C), friend(C,D)}\}$ *and* $\boldsymbol{y}_2 = \{\texttt{friend(A,B), friend(C,D)}\}$*. A global potential function* Φ_f *can compute the degree of friendship of a world assigning a value* $\Phi_f(\boldsymbol{y}_1) = 3$ *to the first world and* $\Phi_f(\boldsymbol{y}_2) = 2$ *to the second world.*

This example gives us the opportunity to provide a first intuition behind our approach. If we compute on our data that the average value of ϕ_f on all the fragments is, let's say, 2.7, then we would like that our model will show an expected value of ϕ_f under the distribution exactly equal to 2.7. We see that the larger and finer these potential functions are, the more our models will be adherent to the observed data (i.e. the larger the number of properties which are required to match the data).

3.2 The Mini-Max Entropy problem in the relational setting

3.2.1 Maximizing the entropy

The Maximum Entropy principle (Jaynes, 1957a,b) is one very interesting candidate to solve the problem of modelling the probability distribution $p(\boldsymbol{y})$ [2]. In this context, it is convenient to consider a possible world $\boldsymbol{y}$ as a vector of *boolean random variables*. The MaxEnt principle states that among all the distributions satisfying a given set of constraints, the only unbiased one is the one that, by making no assumptions on the unknown (unconstrained), is the most informative one; i.e. the one with maximum entropy.

In the scenario of machine learning, the maximum entropy principle is applied by exploiting data to constraint the distribution. In other words, what we ask is to select among all the distributions that "explain" the observed data the one with maximum entropy. The principle of maximum entropy can be seen as a generalization principle, because it gives a way to smoothly (i.e. in an uninformative way) interpolate among observed data and to generalize to unseen data.

Let now state the Maximum Entropy problem in our relational scenario. Remember that our goal is to determine a model for the probability distribution $p(\boldsymbol{y})$ over possible worlds. Suppose we are provided with a single [3] observed world $\hat{\boldsymbol{y}}$.

[2] In the following, when we need to distinguish the true probability distribution from the model probability distribution we will use the explicit subscript p_{DATA} and p_{MODEL}, respectively. When the subscript is not used, we refer to the model distribution.

[3] The same arguments hold in the simpler case in which we are provided with multiple observed worlds. Here we decided to describe the problem with a single observed world in order to underline the validity in the hardest case and to underline the importance to reason about substructures, which will be a fundamental ingredient of our recipe.

The key idea for learning a good distribution w.r.t. the observed data is that we are looking for the maximum entropy distribution which is constrained to have the same statistics of the observed data w.r.t. some properties of interests. To this end, let introduce a collection of *global potentials* Φ_i, that we consider known. Than, we can define a set of constraints:

$$\mathbb{E}_{\hat{\boldsymbol{y}} \sim p_{DATA}}[\Phi_i(\hat{\boldsymbol{y}})] = \mathbb{E}_{\boldsymbol{y} \sim p_{MODEL}}[\Phi_i(\boldsymbol{y})] \quad \forall i$$

where p_{DATA} is the distribution induced by the observed data and p_{MODEL} is the distribution we are learning. In the case of a single observed world $\hat{\boldsymbol{y}}$, the constraints become:

$$\Phi_i(\hat{\boldsymbol{y}}) = \mathbb{E}_{\boldsymbol{y} \sim p_{MODEL}}[\Phi_i(\boldsymbol{y})] \quad \forall i \tag{3.1}$$

As previously introduced and as it will become clear now, this last version of the constraint is intrinsically ill-posed and we must recur to local potentials to constrain our distribution. Indeed, we are constraining a distribution to model worlds with expected values of potentials equal to the value of the potential of a single world, which, clearly, does not represent any valuable statistics to match.

However, by assuming repeated regularities at the fragment level, we could constraint our distribution over fragment-level statistics and not over world-level statistic. This assumption is shared by mostly of the statistical relational learning approaches and it is described by Kuželka et al. (2018) in the setting of maximum entropy models. The idea used is to find a maximum-entropy distribution $p(\boldsymbol{y})$ such that the following two expected values are the same:

- the expected value of $\phi(\gamma)$ where γ is sampled uniformly from $\Gamma_k(\widehat{\boldsymbol{y}})$;
- the expected value of $\phi(\gamma')$ where γ' is sampled uniformly from $\Gamma_k(\boldsymbol{y})$ and $\boldsymbol{y}$ is sampled, in turn, from $p(\boldsymbol{y})$.

The intuition here is that, at least on average, the fragments of the given training example should look similar to the fragments of possible worlds sampled from the distribution.

To achieve such goal, we need to define the constraint in Equation 3.1 in terms of local potentials but we want to still constraint (and model) entire worlds. To this end, we can factorize a global potential as an aggregation function $\mathcal{A}$ of local potentials.

$$\Phi(\boldsymbol{y}) = \mathcal{A}_{\gamma \in \Gamma_k(\boldsymbol{y})}(\phi(\gamma)) \tag{3.2}$$

where $\mathcal{A}_{a \in A}$ is a function aggregating its arguments evaluated on all the elements a of a set A. We will come back to aggregation function in the next chapter where we will see that multiple choices are possible. For the sake of this chapter, it will be sufficient to think at the most common aggregation function, i.e. the average operator:

$$\Phi(\boldsymbol{y}) = \frac{1}{|\Gamma_k(\boldsymbol{y})|} \sum_{\gamma \in \Gamma_k(\boldsymbol{y})} \phi(\gamma)$$

Now that we know how to constraint our problem, even when only a single observed world is available, we can exploit the maximum entropy principle to state an optimization problem guiding the learning of the target distribution.

For the sake of simplicity and without loss of generality, let suppose the set $\mathcal{Y}$ of all possible worlds to be discrete so that the function $p(\boldsymbol{y})$ can be substituted with a set of $|\mathcal{Y}|$ variables $P_{\boldsymbol{y}}$. Let us also assume that there exists at least one distribution $P_{\boldsymbol{y}}$ which is a model of the true distribution, i.e. the problem admits at least one solution. Then, the MaxEnt problem is stated as follow:

$$\max_{P_y} - \sum_{\boldsymbol{y} \in \mathcal{Y}} P_{\boldsymbol{y}} \log(P_{\boldsymbol{y}}) \tag{3.3}$$

$$\text{s.t. } (1) \sum_{\boldsymbol{y} \in \mathcal{Y}} P_{\boldsymbol{y}} = 1; \tag{3.4}$$

$$(2)\ \Phi_i(\hat{\boldsymbol{y}}) = \mathbb{E}_{\boldsymbol{y} \sim P_{\boldsymbol{y}}}[\Phi_i(\boldsymbol{y})] \quad \forall i \tag{3.5}$$

where we are looking for the distribution values $P_{\boldsymbol{y}}$ such that:

- the probability distribution has maximum entropy (Equation 3.3);
- it behaves like a probability (sums to 1)[4] (Equation 3.4);
- the statistics of the given potentials on the samples from the distribution match those on the observed data (Equation 3.5) .

As commonly done, we refer to this problem as the *primal* problem.

[4]In principle, a non negative constraint, $P_{\boldsymbol{y}} \geq 0\ \forall \boldsymbol{y}$, should also be added to enforce probabilities to be positive. However, the particular choice of Shannon entropy and its logarithmic shape implicitly restrict the domain of the $P_{\boldsymbol{y}}$ to $\mathbb{R}^+$ and, thus, implicitly enforce this constraint. This result will also holds for any measure of the class $dist(p, p_0) = \sum_i p_i \phi(p_{0,i}/p_i)$ under linear constraints will satisfy the result of $p_i \geq 0$ provided that ϕ^{-1} is an exponential. $H(p) = KL(p||uniform)$ is a member of this class of measures.

3.2.2 The dual problem

In this section we show how we can move from the hard primal constrained problem to a dual easier unconstrained problem. We can solve this problem using standard convex optimization argument (Boyd and Vandenberghe, 2004). To this end, we need to construct the Lagrangian $L(P, \beta, z)$ of the maximum entropy problem, where we define P the vector of all $P_{\boldsymbol{y}}$, β the vector of all the Lagrange multiplier β_i for the constraints in Equation 3.5 and z the Lagrange multiplier for the constraint in Equation 3.4 . Moreover, let us omit the non negativity constraints as it will turn out that they are enforced implicitly.

The Lagrangian corresponding to the MaxEnt problem is:

$$L(P, \beta, z) = -\sum_{\boldsymbol{y} \in \mathcal{Y}} P_{\boldsymbol{y}} \log(P_{\boldsymbol{y}}) + \sum_i \beta_i \Big(\sum_{\boldsymbol{y} \in \mathcal{Y}} \Phi_i(\boldsymbol{y}) P_{\boldsymbol{y}} - \Phi_i(\widehat{\boldsymbol{y}}) \Big) + z \Big(1 - \sum_{\boldsymbol{y} \in \mathcal{Y}} P_{\boldsymbol{y}} \Big) \tag{3.6}$$

and the optimization problem is translated into the unconstrained maximization of the Lagrangian:

$$\max_{P, \beta, z} L(P, \beta, z)$$

Now, holding β fixed, to find the stationary points of $L(P, \beta, z)$ w.r.t. P, we take the partial derivatives of the Lagrangian w.r.t. $P_{\boldsymbol{y}}$:

$$\frac{\partial}{\partial P_{\boldsymbol{y}}} L(P, \beta, z) = -\log P_{\boldsymbol{y}} + \sum_i \beta_i \Phi_i(\boldsymbol{y}) - z + 1$$

and set them equal zero:

$$-\log P_{\boldsymbol{y}} + \sum_i \beta_i \Phi_i(\boldsymbol{y}) - z + 1 = 0$$

so we obtain:

$$P_{\boldsymbol{y}} = \exp(\sum_i \beta_i \Phi_i(\boldsymbol{y})) \exp(1 - z)$$

By substituting this equation into the stationary point w.r.t. z, which is recovering the normalization constraint, we obtain that $\exp(1 - z)$ is simply the normalization

factor. So we can state that, the solution to the MaxEnt problem, if exists, must have the following form:

$$P_{\boldsymbol{y}} = \frac{1}{Z_\beta} \exp(\sum_i \beta_i \Phi_i(\boldsymbol{y})) \tag{3.7}$$

with Z_β being the *normalization factor* (or *partition function*):

$$Z_\beta = \sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(\sum_i \beta_i \Phi_i(\boldsymbol{y})) \tag{3.8}$$

So by analytical arguments, we showed that the MaxEnt problem has solutions of the exponential form of Equation 3.7. By substituting the optimal P of Equation 3.7 (which are functions of the only unknown β) into Equation 3.6 , we obtain the *dual* problem as:

$$\max_\beta L(\beta)$$

where, by some algebraic manipulation:

$$L(\beta) = \sum_i \beta_i \Phi_i(\widehat{\boldsymbol{y}}) - \sum_{\boldsymbol{y} \in \mathcal{Y}} \log Z_\beta \tag{3.9}$$

Due to the positivity assumption, Slater's condition (Boyd and Vandenberghe, 2004) is satisfied and strong duality holds.

By moving to the dual problem, we have obtained multiple advantages that we describe in turn:

- **Reduction of parameter search space:** indeed, we moved from a problem, the primal, in which we needed to optimize $n = |\mathcal{Y}|$ values $P_{\boldsymbol{y}}$, to another problem, the dual, where we need to optimize only $m \ll |\mathcal{Y}|$ values β_i. Here m is the number of features, and thus of constraints, we want our model distribution to match the real distribution (i.e. $0 \leq i < m$)

- **Maximum Log-Likelihood duality:** by a carefully inspection, it is immediately verifiable that the simplified Lagrangian in Equation 3.9 is the log-likelihood of the optimal distribution $P_{\boldsymbol{y}}$ of Equation 3.7 w.r.t the observed world $\widehat{\boldsymbol{y}}$.

3.2.3 The correspondent Markov Random Field

The functional shape of the solution $p(\boldsymbol{y})$ is clearly a distribution from the family of exponential distribution. This allows us to interpret the correspondent model in terms of a Markov Random Field (MRF), i.e. a probabilistic undirected graphical model.

Here, the random variables under investigations are the truth values of the possible world $\boldsymbol{y}$, which are then treated as *boolean random variables*. Sometimes, we need to relax this discreteness and consider the $\boldsymbol{y}$ as continuous random variables in the real interval $[0, 1]$ (see Chapter 6).

The potentials $\Phi(\boldsymbol{y})$ clearly represents factors of the corresponding MRF. We show a schematic view of the defined MRF in Figure 3.2.

It is interesting to analyse the role of fragmentation in terms of factorization. Indeed, potentials in an undirected probabilistic graphical model are individuating a conditional independence scheme. All the variables of a potential[5] are conditionally dependent only on the other variables belonging to the same potential. This independence is easily verifiable as a factorization of the distribution function:

$$p(\boldsymbol{y}) = \frac{1}{Z} \exp\left(\beta_i \sum_i \Phi_i(\boldsymbol{y})\right) = \frac{1}{Z} \prod_i \exp\left(\beta_i \Phi_i(\boldsymbol{y})\right)$$

When a potential is modelled as an aggregation over fragments (see Equation 3.2), it is worth noticing that some aggregation scheme (e.g. sum, mean) increase the degree of factorization of the corresponding MRF. In particular, factorizing the potential $\Phi(\boldsymbol{y})$ as a sum over local potentials $\phi(\gamma)$ on fragments γ of $\boldsymbol{y}$ yields:

$$p(\boldsymbol{y}) = \frac{1}{Z} \prod_i \exp\left(\beta_i \Phi_i(\boldsymbol{y})\right) = \frac{1}{Z} \prod_i \prod_{\gamma \in \Gamma_k(\boldsymbol{y})} \exp\left(\beta_i \phi_i(\gamma)\right)$$

This is extremely beneficial from an inference viewpoint since exact inference algorithms usually scale polynomially with the number of the factors and exponentially with the size of the factors. Thus fragmenting the distribution into a larger number of smaller factors allows a much easier inference.

3.2.4 Minimizing the Entropy

Potential Learning. In Section 3.2.1, we have seen that, given some statistics on a given set of potentials on the training data, the MaxEnt principle allows us to select the most simple distribution matching those statistics. This phase is often

[5]It is usually the case that potentials are function of only a subset of the possible world $\boldsymbol{y}$

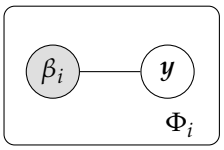


Figure 3.2: The graphical model corresponding to the Max Entropy model in Equation 3.7.

referred as *feature binding* because given a set of features (i.e. potentials) we are *binding* a distribution to these statistics. As we will see in the following Sections, when potentials are given First-Order-Logic formulas, the MaxEnt model recovers a very well known statistical relational approach, known as Markov Logic Network.

Even though MaxEnt models in the relational setting can be very useful in many contexts (see (Richardson and Domingos, 2006)), their potential is limited in the fact that they need to be provided with a set of potentials in advance. Thus, in the use of these models, domain experts are required to design some useful features about the domain of interest by hand (e.g. logical rules) or structure learning based on combinatorial search needs to be performed. These requirements normally limit a wide application of these models as out-of-the box tools. It is worth noticing that overtaking the need of such "feature-engineering" is one of the reasons behind the massive adoption of deep learning techniques.

In this Section, we remove the assumption of the potentials to be given and we will introduce a complementary principle where we would like to select the best features by minimizing the entropy of the model probability distribution. This second phase is usually referred to as *feature selection*, since we want to learn which are the features that mostly characterize our observed data.

We will now provide the reader with an intuitive explanation of the minimization principle that will be followed by a technical explanation

The minimum entropy principle. To the sake of this explanation, suppose we have the MaxEnt problem defined of Equation 3.3, but without constraints on the statistics of potentials. It is easily verifiable that the only solution is the uniform distribution, i.e. the distribution with maximum entropy among all the probability distributions.

Now, suppose we add a single constraint on the statistics using a potential $\Phi(\boldsymbol{y})$. If this potential is informative and makes some worlds more likely than others, then the solution moves from the uniform distribution to another distribution with *lower* entropy. The more the feature is discriminative (i.e. informative) of the worlds (or fragments) in the observed data, the more the reduction of the entropy will be. In terms of information theory, the more the feature is discriminative of the data the less our surprise when sampling a world from the distribution.

A very intuitive way of interpreting the potentials, according to this qualitative

justification, is that the set of potentials can be considered a *language* we provide to our MaxEnt optimization problem to describe the data we are analysing. The richer the language the better the distribution we derive. Recovering basic concepts from information theory (Shannon, 1948; Wiegand et al., 2011; Cover and Thomas, 2012), the entropy of a probabilistic source measures the minimum average bit-length needed in order to compress the data generated by that source in a lossless manner. Thus, in the selection of the potentials, which concur in modelling the *probabilistic source* of the data we are observing, the minimum entropy distribution is the one which recovers the most lossless compressive potentials.

So, the need for a principle to select the best feature to describe our data bring us to selecting the features that minimize the most the entropy of the model distribution.

There is a interesting technical reason (Zhu et al., 1997) behind the minimization of the entropy that we describe in turn. For the sake of simplicity in the proof, suppose that we are provided with multiple worlds so that the constraint of statistics explicitely show the underlying data probability distribution; i.e.:

$$\mathbb{E}_{P_{\text{DATA}}}[\Phi_i(\boldsymbol{y})] = \mathbb{E}_{P_{\text{MODEL}}}[\Phi_i(\boldsymbol{y})], \qquad 0 \leq i < m \tag{3.10}$$

Since our goal is to make an inference about the underlying data distribution P_{DATA}, we can measure the goodness of the current model P_{MODEL} using the Kullback-Leibler divergence (Kullback and Leibler, 1951):

$$\begin{aligned} KL(P_{\text{DATA}}||P_{\text{MODEL}}) &= \sum_{\boldsymbol{y}\in\mathcal{Y}} P_{\boldsymbol{y},\text{DATA}} \log \frac{P_{\boldsymbol{y},\text{DATA}}}{P_{\boldsymbol{y},\text{MODEL}}} \\ &= \mathbb{E}_{P_{\text{DATA}}}[\log P_{\boldsymbol{y},\text{DATA}}] - \mathbb{E}_{P_{\text{DATA}}}[\log P_{\boldsymbol{y},\text{MODEL}}] \\ &= -H(P_{\text{DATA}}) - \mathbb{E}_{P_{\text{DATA}}}[\log P_{\boldsymbol{y},\text{MODEL}}] \end{aligned}$$

where $H()$ is the entropy of the input probability distribution.

Let us perform some algebraic manipulations of the second term of the subtraction:

$$\begin{aligned} \mathbb{E}_{P_{\text{DATA}}}[\log P_{\boldsymbol{y},\text{MODEL}}] &= \sum_i \mathbb{E}_{P_{\text{DATA}}}[\beta_i \Phi_i(\boldsymbol{y})] - \mathbb{E}_{P_{\text{DATA}}}[\log Z] \\ &= \sum_i \beta_i \mathbb{E}_{P_{\text{DATA}}}[\Phi_i(\boldsymbol{y})] - \log Z \end{aligned}$$

Now, by using the solution of the maximum entropy problem, we know that the constraints on the expectations (Equation 3.10), whatever the constraints look like, are satisfied. By using this result, we have:

$$\begin{aligned}\mathbb{E}_{P_{\text{DATA}}}[\log P_{\boldsymbol{y},\text{MODEL}}] &= \sum_i \beta_i \mathbb{E}_{P_{\text{DATA}}}[\Phi_i(\boldsymbol{y})] - \log Z \\ &= \sum_i \beta_i \mathbb{E}_{P_{\text{MODEL}}}[\Phi_i(\boldsymbol{y})] - \log Z \\ &= \mathbb{E}_{P_{\text{MODEL}}}[\log P_{\boldsymbol{y},\text{MODEL}}] \\ &= -H(P_{\text{MODEL}})\end{aligned}$$

Finally, we have shown what is reported in the Theorem 2.

Theorem 2 (KL divergence for maximum entropy models)**.** *Given a data distribution P_{DATA} and a target model distribution P_{MODEL}, which is a solution to the maximum entropy problem of Equation 3.3, we have:*

$$KL(P_{DATA}||P_{MODEL}) = H(P_{MODEL}) - H(P_{DATA})$$

Theorem 2 provides us with another viewpoint in justifying the principle of minimum entropy. Indeed, it is evident that, by minimizing the entropy of the model distribution, i.e. $H(P_{\text{MODEL}})$, we are actually minimizing the KL divergence between the underlying true distribution represented by our data samples and the distribution we are modelling.

Corollary 1 (Equivalence of the minimization of entropy and KL divergence)**.** *If P_{MODEL} is the solution of the MaxEnt problem in Equation 3.3 under constraints in Equations 3.4 and 3.5, then, it follows from Theorem 2 that the minimization of its entropy $H(P_{MODEL})$ is equivalent to the minimization of the KL Divergence between this distribution and the true data distribution it has been constrain on in Equation 3.5.*

3.2.5 The mini-max entropy problem

In the previous sections, we have shown two interleaved principles for learning in the relational setting, namely maximum and minimum entropy. Now, we try to put them together in a uniform definition. We recall our goal of finding a probability distribution $P_{\boldsymbol{y}}$ that models the true, unknown, probability distribution represented by some observed data samples.

In order to allow us to proceed in the feature selection enabled by the minimum entropy principle, we need to introduce a new class of *parametric fragment potential functions*, $\phi(\gamma; \boldsymbol{w})$, and their correspondent *parametric global potential function*:

$$\Phi(\boldsymbol{y}; \boldsymbol{w}) = |\Gamma_k(\boldsymbol{y})|^{-1} \sum_{\gamma \in |\Gamma|} \phi(\gamma; \boldsymbol{w}) \tag{3.11}$$

.

Here, $\boldsymbol{w}$ represents a set of parameters controlling the behaviour of ϕ. Thus, selecting the correct features turns in selecting the right set of parameters $\boldsymbol{w}$.

We have seen in Section 3.2.2, that we can move from maximizing the entropy w.r.t. the probabilities $P_{\boldsymbol{y}}$ to the maximization of the likelihood w.r.t. a new set β of parameters controlling the distribution $P_{\boldsymbol{y}}$, under the exponential form in Equation 3.7, with fixed potentials. Let us define $H(\beta)$ the entropy of the distribution $P_{\boldsymbol{y}}$ controlled by the parameters β. The primal problem can, thus, be reframed as: $\max_\beta H(\beta)$ subject to constraints in Equations 3.5 and 3.4. Up to this step, we are facing only the problem of *features binding*; i.e. finding the best probability given some fixed potentials.

Let us introduce parametric potentials $\Phi(\boldsymbol{y}; \boldsymbol{w})$ on the distribution $P_{\boldsymbol{y}}$, which now depends both on β and on $\boldsymbol{w}$. The corresponding entropy is $H(\beta, \boldsymbol{w})$. Thus, we can improve our optimization problem by adding a *feature selection* capability by asking that the same entropy should be minimized w.r.t. the parameters $\boldsymbol{w}$ controlling the behaviour of the potentials.

Finally, we obtain our final problem of *Mini-Max Entropy*:

$$\begin{aligned} &\min_{\boldsymbol{w}} \max_{\beta} H(\beta, \boldsymbol{w}) \\ &\quad \text{s.t. } (1) \sum_{\boldsymbol{y} \in \mathcal{Y}} P_{\boldsymbol{y}} = 1, P_{\boldsymbol{y}} \geq 0 \forall \boldsymbol{y}; \\ &\qquad (2)\ \Phi_i(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \mathbb{E}_{\boldsymbol{y}, \boldsymbol{w} \sim P_{\boldsymbol{y}}}[\Phi_i(\boldsymbol{y}, \boldsymbol{w})] \quad \forall i \end{aligned} \tag{3.12}$$

Let us now fix the value of $\boldsymbol{w}$. We can then tackle the maximization problem and, exploiting the duality arguments of Section 3.2.2, we can state that the maximization of the entropy is equivalent to maximization of the log-likelihood, $\max \log P_{\boldsymbol{y}}$, or equivalently to the minimization of the negative log-likelihood. Let's call the solution of the MaxEnt problem with a particular value of $\boldsymbol{w}$, $P^\star_{\boldsymbol{y}}(\boldsymbol{w})$, and the corresponding set of optimal parameters $\beta^\star$. In other words:

$$\beta^\star = \arg\min_{\beta} -\log P_{\boldsymbol{y}}(\boldsymbol{w}) \tag{3.13}$$

$$\text{with } \ P_{\boldsymbol{y}}(\boldsymbol{w}) = \frac{1}{Z} \exp \sum_i \beta_i \Phi(\boldsymbol{y}; \boldsymbol{w}) \tag{3.14}$$

Now, we can fix the parameters $\beta = \beta^\star$ and substitute this solution to the problem in Equation 3.12, yielding:

$$\min_{\boldsymbol{w}} \min_{\beta} -\log P_{\boldsymbol{y}} \tag{3.15}$$

which is clearly equivalent to:

$$\max_{\beta, \boldsymbol{w}} \log P_{\boldsymbol{y}}$$

Thus we have an important result. The MiniMax Entropy problem is still equivalent in maximizing the likelihood w.r.t. the set of both parameters β and $\boldsymbol{w}$.

Finally, we want to give an intuition behind the coupled minimization and maximization of Entropy, in terms of development principle for an intelligent agent. Let us resume both principles:

- The Maximization of Entropy is a principle that, among all the distributions that satisfy a set of constraints, selects the most entropic one, that is the one that makes less assumptions. In our setting, constraints are enforcing some features to be equally distributed in the data and in the model. Thus, the entropy here can be seen as a measure of confusion of an agent willing to describe the observations of its environment. Given the current features, the most messy explanation of the data is the one provided by the MaxEnt distribution. This principle is selecting the *worst-case* distribution (in terms of confusion), which is the simplest one and the one that requires no more assumptions.

- The Minimization of Entropy instead tries to select the most relevant features to describe the data, thus minimizing the confusion of the artificial agent.

The MiniMax Entropy problem can thus seen in terms of the minimization of the maximum confusion the agent can reach. We can rephrase the problem as follows: *find the distribution which minimize the maximum confusion of the agent.*

3.2.6 Estimation and Computation

We have seen that the Mini-Max Entropy problem has again a Maximum Likelihood problem as equivalent problem. A gradient based scheme would exploit gradients w.r.t. the trainable parameters. Thus, let compute this derivatives first and, then, describe the corresponding gradient ascent optimization.

Let $w_{i,j}$ the single j-th parameter of the i-th potential and β_i the dual parameter associated to the i-th constraint in Equation 3.12.

A gradient-based scheme will require to find a stationary point of the likelihood w.r.t. to both these set of parameters. Then, let us make the derivative w.r.t. $w_{i,j}$ and β_i.

$$\frac{\partial \log(P_{\widehat{\boldsymbol{y}}})}{\partial w_{i,j}} = \beta_i \left(\frac{\partial \Phi_i(\widehat{\boldsymbol{y}}; \boldsymbol{w}_i)}{\partial w_{i,j}} - \mathbb{E}_{\boldsymbol{y} \sim P} \left[\frac{\partial \Phi(\boldsymbol{y}; \boldsymbol{w}_i)}{\partial w_{i,j}} \right] \right) \tag{3.16}$$

$$\frac{\partial \log(P_{\widehat{\boldsymbol{y}}})}{\partial \beta_i} = \Phi_i(\widehat{\boldsymbol{y}}; \boldsymbol{w}_i) - \mathbb{E}_{\boldsymbol{y} \sim P} \left[\Phi(\boldsymbol{y}; \boldsymbol{w}_i) \right] \tag{3.17}$$

Some comments of these derivatives are in order:

- Equation 3.17, at stationary conditions, recovers the original constraints in Equation 3.5.
- Equation 3.16 is a consequence of the minimization of the entropy and requires that, at stationary conditions, not only the values of the potentials should match, but also their first derivatives. This could be an interesting alternative viewpoint w.r.t. recent approaches in Sobolev Training in neural networks using derivable target functions (Czarnecki et al., 2017). When the Φ functions are neural functions (as it will be always in the rest of this dissertation), then the derivatives on the right-hand side of the Equation will trigger a Backpropagation step.
- Both the derivatives require the computation of an expectation w.r.t. the target model probability distribution. Thus, the gradient descent scheme goes implicitly in the Expectation Maximization direction. In order to perform a maximization gradient step, we need to compute an expectation over the current model, which means we need to infer from the model the expected value of the potential functions. For any but trivial problems, an exhaustive computation of this expected value is infeasible and one must recover to approximate inference methods as we will show in the next Section.

Before going deeper into questions regarding inference in probabilistic models, it is useful to depict a general algorithm (Algorithm 1) for the maximization of the likelihood in a Mini-Max problem. Most of the steps of this algorithm will be placeholders for specific strategies that we will propose in the rest of this thesis.

It interesting to note the *EM* structure of this algorithm: the computation of some expectations is required for each maximization step (i.e. updates of parameters). In particular, this algorithm retraces more a special case of *EM*, i.e. *coordinate gradient ascent* for latent variable models, since the maximization step is not carried out completely but only one update of the parameters per expectation.

3.2.7 A review of approximate inference in probabilistic models

In Section 7.2.1 we discussed how the computation of the gradients w.r.t. the distribution parameters requires the computation of an expected value under the actual distribution, whose exact computation is, in most of the interesting cases, infeasible.

Data: Input data $\widehat{\boldsymbol{y}}$, potentials (parametric) functions Φ
Result: Trained model parameters $\boldsymbol{\beta}, \boldsymbol{w}$
Initialize $k = 0$, $\boldsymbol{\beta} = \mathbf{0}$, random $\boldsymbol{w}$;
while *not converged* $\wedge$ $k <$ *max_iterations* **do**
  forall *potential index* i **do**
    Compute potentials on data $\Phi_i(\widehat{\boldsymbol{y}}; \boldsymbol{w}_i)$;
    Compute potentials gradients on data $\nabla_{\boldsymbol{w}_i} \Phi_i(\widehat{\boldsymbol{y}}; \boldsymbol{w}_i)$;
    Compute potentials expected values $\mathbb{E}[\Phi(\boldsymbol{y}; \boldsymbol{w}_i)]$;
    Compute potentials gradients expected values $\mathbb{E}[\nabla_{\boldsymbol{w}_i} \Phi_i(\boldsymbol{y}; \boldsymbol{w}_i)]$;
    Compute log-likelihood gradient $\nabla_{\boldsymbol{w}_i} \log(P_{\widehat{\boldsymbol{y}}})$
    Compute log-likelihood gradient $\nabla_{\beta_i} \log(P_{\widehat{\boldsymbol{y}}})$
    Update $\boldsymbol{w}_i$ via gradient ascent: $\boldsymbol{w}_i^{k+1} = \boldsymbol{w}_i^k + \lambda_{lr} \cdot \nabla_{\boldsymbol{w}_i} \log(P_{\widehat{\boldsymbol{y}}})$;
    Update β_i via gradient ascent: $\beta_i^{k+1} = \beta_i^k + \lambda_{lr} \cdot \nabla_{\beta_i} \log(P_{\widehat{\boldsymbol{y}}})$;
  end
  Set $k = k + 1$;
end

Algorithm 1: Iterative gradient ascent algorithm to train a Mini-Max entropy model parametrized by the training parameters $\boldsymbol{w}_i$ and β_i.

In this Section, we review two macro approaches to make approximate inference, in particular Markov-Chain Monte Carlo and Variational Inference. Both these methods will be exploited up to a certain degree in the following chapters.

Markov-Chian Monte Carlo When a sum or an integral cannot be computed exactly (for example, the sum has an exponential number of terms, and no exact simplification is known, as it is the case in most of the application of the proposed model), it is often possible to approximate it using Monte Carlo sampling. The idea is to approximate the expectation by the corresponding empirical average over samples from the distribution.

$$\mathbb{E}[f(x)] = \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

where we have exploited N samples x_i from the distribution for the sake of the approximation. This can be showed to be an unbiased estimator for the expected value. Moreover, the law of large numbers ensure that, if the samples are i.i.d., than the correct expected values is recovered in the limit $N \to \infty$ (Robert and Casella, 2013).

Clearly, Monte Carlo sampling has simply changed the problem from computing expectation to sampling from the distribution. When we don't have ways of directly sampling from the distribution, a general approach is to form a sequence of estimators

that converge toward the distribution of interest. This is known as Markov-Chain Monte Carlo methods. The core idea of a Markov chain is to have a state x that begins as an arbitrary value. Over time, we randomly update x repeatedly. Eventually, x becomes (very nearly) a fair sample from $p(x)$. Formally, a Markov chain is defined by a random state x and a transition distribution $T(x'|x)$ specifying the probability that a random update will go to state x' if it starts in state x. Running the Markov chain means repeatedly updating the state x to a value x' with probability $T(x'|x)$. When we have a countable finite number of states we can consider x the index of the corresponding state and the transition probability can be analysed in terms of a transition matrix. Running multiple times the chain will solve a fixed point equation of the transition matrix and, by correctly choosing the T probability, the corresponding fixed point is indeed our target distribution and we are sampling from it.

Variational Inference Another way of making (exact) inference in probabilistic models is to treat the inference problem as an optimization problem. This will allow to easily find new ways of approximating the inference as approximations to the correspondent optimization process.

The general idea is to introduce a set of latent variables, z, and to reason about the joint distribution $p(\boldsymbol{y}, z; \boldsymbol{\beta}, \boldsymbol{w})$ instead of the marginal $p(\boldsymbol{y}; \boldsymbol{\beta}, \boldsymbol{w})$. In particular, one will evaluate the conditional probability $p(\boldsymbol{y}|z; \boldsymbol{\beta}, \boldsymbol{w})$ and it recovers the joint by exploiting the relation $p(\boldsymbol{y}, z; \boldsymbol{\beta}, \boldsymbol{w}) = p(z; \boldsymbol{\beta}, \boldsymbol{w}) p(\boldsymbol{y}|z; \boldsymbol{\beta}, \boldsymbol{w})$ and the marginal $p(\boldsymbol{y}; \boldsymbol{\beta}, \boldsymbol{w})$ by marginalizing the joint distribution w.r.t. z. For the sake of simplicity in the notation, let call all the parameters of the model, i.e. $\boldsymbol{\beta}$ and $\boldsymbol{w}$, simply as θ.

Another hypothesis, which will be easily explainable in exponential family models, like our MiniMax entropy model, is that the direct optimization of the log-likelihood $p(\widehat{\boldsymbol{y}}; \theta)$ is difficult, but that the optimization of the complete-data likelihood function $p(\widehat{\boldsymbol{y}}, z; \theta)$ is much easier.

Then we introduce a new distribution $q(z)$ defined over the latent variables only. For *any choice* of $q(z)$, the following decomposition holds:

$$\log p(\widehat{\boldsymbol{y}}; \theta) = \mathcal{L}(q; \theta) + KL(q||p)$$

where $KL(q||p)$ is the Kullback-Leibler divergence between $q(z)$ and the posterior distribution $p(z|x; \theta)$, while $\mathcal{L}(q; \theta)$ is defined as:

$$\mathcal{L}(q; \theta) = \sum_z q(z) \log \left(\frac{p(\boldsymbol{y}, z; \theta)}{q(z)} \right)$$

Since, by definition, $KL(q||p) \geq 0$, then $\mathcal{L}(q; \theta) \leq \log p(\widehat{\boldsymbol{y}}; \theta)$. In other words, $\mathcal{L}(q; \theta)$ is a **lower bound** on $\log p(\widehat{\boldsymbol{y}}; \theta)$. Thus, we can maximize the lower bound

$\mathcal{L}(q;\theta)$ with respect to the distribution $q(z)$, which is equivalent to minimizing the KL divergence. If we allow any possible choice for $q(z)$, then the maximum of the lower bound occurs when the KL divergence vanishes, which occurs when $q(z)$ equals the posterior distribution $p(z|\boldsymbol{y};\theta)$.

However, we shall suppose the model is such that working with the true posterior distribution is intractable. We therefore consider instead a restricted family of distributions $q(z)$ and then seek the member of this family for which the KL divergence is minimized. Our goal is to restrict the family sufficiently that they comprise only tractable distributions, while at the same time allowing the family to be sufficiently rich and flexible that it can provide a good approximation to the true posterior distribution. More details on classical variational inference in Bishop (2006).

3.3 The conditional case

In the previous Section, we introduced and discussed the principle of Mini-Max Entropy and its corresponding problem in the case of a standard relational setting, where ground atoms are simply boolean random variables, expressing symbolic knowledge. Modelling probability distribution over these symbolic worlds, as described in the previous Sections, enables different symbolic reasoning tasks to be designed. However as discussed in Chapter 1, neuro-symbolic approaches shine in their capability to merge reasoning at the symbolic level with reasoning at the subsymbolic (or perceptual) level. In this section we are going to enhance our approach to foresee both symbolic reasoning and subsymbolic reasoning, where perceptual features of the constants[6] are exploited.

The simple key idea to move to the conditional case is that the model probability distribution $p(\boldsymbol{y})$[7] is substituted with a conditional probability distribution $p(\boldsymbol{y}|x_{\boldsymbol{y}})$. Here $x_{\boldsymbol{y}}$ is the collection of feature representations of constants; i.e. $x_{\boldsymbol{y}} = \{g(c)\ \forall c \in C\}$, where C is the constant set of the given world $\boldsymbol{y}$. In the following, we will drop the subscript $_{\boldsymbol{y}}$ in $x_{\boldsymbol{y}}$ when it is clear from the context that x is referred to the current evaluated world $\boldsymbol{y}$. In terms of probabilistic graphical models, the random variable x is an observed variable which we don't want to infer about but we want to exploit in the inference about ground atoms of $\boldsymbol{y}$.

A fundamental ingredient in the extension to the conditional case is the definition of a new class of potentials which are able to exploit also the perceptual representation of the constants. In particular, we define *conditional (parametric) global potential function* $\Phi(\boldsymbol{y}|x_{\boldsymbol{y}};\boldsymbol{w})$ and its corresponding *conditional (parametric) local potential functions* $\phi(\gamma|x_\gamma;\boldsymbol{w})$.

[6] In principle, also features of the relations can be modeled in the same way but we do not consider this casuistry in this thesis.

[7] In this section we will exploit again a continuous probability distribution.

An important question that need to be answered in order to extend our model to the conditional case is: *how can we constraint the statistics of the potentials of our model?*. Indeed, suppose we have a dataset $\mathcal{D}$ of pairs $(\hat{x}, \hat{\boldsymbol{y}})$. They are samples of the joint distribution $p_{\text{DATA}}(\boldsymbol{y}, x)$, which we suppose to be too difficult to be modeled directly. Thus, constraints can only be expressed on this joint distribution in the following way:

$$\mathbb{E}_{p_{\text{DATA}}(\boldsymbol{y},x)}[\Phi(\boldsymbol{y}|x;\boldsymbol{w})] = \mathbb{E}_{p_{\text{MODEL}}(\boldsymbol{y},x)}[\Phi(\boldsymbol{y}|x;\boldsymbol{w})]$$
$$\frac{1}{|\mathcal{D}|}\sum_{(\hat{x},\hat{\boldsymbol{y}})\in\mathcal{D}}\Phi_i(\hat{\boldsymbol{y}}|\hat{x};\boldsymbol{w}) = \sum_{x}\sum_{\boldsymbol{y}\in\mathcal{Y}}\Phi(\boldsymbol{y}|x;\boldsymbol{w})p_{\text{MODEL}}(\boldsymbol{y},x)$$

But we still want to use our data to constraint the conditional distribution. We can express the joint distribution as $p(\boldsymbol{y}, x) = p(\boldsymbol{y}|x)p(x)$ and, since we are not interested in modeling $p(x)$, we can use the observed data to have an approximation of $p(x)$. Thus:

$$\frac{1}{|\mathcal{D}|}\sum_{(\hat{x},\hat{\boldsymbol{y}})\in\mathcal{D}}\Phi_i(\hat{\boldsymbol{y}}|\hat{x};\boldsymbol{w}) = \frac{1}{|\mathcal{D}|}\sum_{\hat{x}\in\mathcal{D}}\sum_{\boldsymbol{y}\in\mathcal{Y}}\Phi(\boldsymbol{y}|\hat{x};\boldsymbol{w})p_{\text{MODEL}}(\boldsymbol{y}|\hat{x})$$

Finally, we are ready to express the Mini-Max Entropy principle in the conditional case as:

$$\min_{\boldsymbol{w}}\max_{\beta} H(\beta,\boldsymbol{w}) \tag{3.18}$$
$$\text{s.t. } (1)\ \sum_{\boldsymbol{y}\in\mathcal{Y}} p(\boldsymbol{y}|x) = 1, p(\boldsymbol{y}|x) \geq 0\ \forall \boldsymbol{y};$$
$$(2)\ \frac{1}{|\mathcal{D}|}\sum_{(\hat{x},\hat{\boldsymbol{y}})\in\mathcal{D}}\Phi_i(\hat{\boldsymbol{y}}|\hat{x};\boldsymbol{w}) = \frac{1}{|\mathcal{D}|}\sum_{\hat{x}\in\mathcal{D}}\sum_{\boldsymbol{y}\in\mathcal{Y}}\Phi_i(\boldsymbol{y}|\hat{x};\boldsymbol{w})p(\boldsymbol{y}|\hat{x})$$

The same arguments of the non conditional case hold and the solution to the mini-max entropy problem is also the solution to an equivalent maximum likelihood problem for models of the same exponential form.

3.3.1 MAP Inference

We usually use the term *inference* to refer to computing the probability distribution over one set of variables given another. When training probabilistic models with latent/output variables, we are usually interested in computing $p(\boldsymbol{y}|x;\theta)$, where, again, θ refers to the set of parameters $\boldsymbol{\beta}$ and $\boldsymbol{w}$.

An alternative form of inference is to compute the single most likely value of the missing variables, rather than to infer the entire distribution over their possible values. In the context of latent variable models, this means computing

$$\boldsymbol{y}^\star = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|x;\theta)$$

This is known as maximum a posteriori inference, abbreviated as *MAP inference*.

By looking for a single value instead of inferring an entire distribution, MAP inference in some cases can be an easier task than full inference, even though it is not always the case. It is interesting to refer to one of these cases since, in Chapter 6, it will be an important ingredient to make the learning procedure faster. In particular, let us have a look at a generic exponential family distribution, like the Mini-Max entropy models introduced in this chapter:

$$p(x) = \frac{e^{\phi(x)}}{\sum_{\hat{x}\in\mathcal{X}} e^{\phi(\hat{x})}}$$

A MAP inference task would look for the single $x^\star$ which maximises $p(x)$. However, by summing over all the possible values of $\hat{x}$, the partition function in the denominator is not a function of x. Thus maximising the $p(x)$ is equal to maximizing the numerator only, and, in particular, only the exponent, since the exponential is a strictly increasing monotonic function. Thus:

$$x^\star = \arg\max_x p(x) = \arg\max_x \phi(x)$$

In this case, we can exploit some features of ϕ function to make the MAP inference faster. For example, for differentiable and convex ϕ function, the MAP inference task is extremely fast if compared with naive approaches that would require an exhaustive search in the variable space.

3.3.2 Bridging symbolic and subsymbolic reasoning

In this Section, we are going to give the reader an intuition about the way MiniMax Entropy models in the conditional case could represent a bridge between symbolic reasoning and subsymbolic reasoning.

To this end, let us look more carefully at the general shape of the conditional global potential function $\Phi(\boldsymbol{y}|x_{\boldsymbol{y}};\boldsymbol{w})$. It is a function which should assign an higher potential value to those configurations of $\boldsymbol{y}$ which are more representative of the data they are learned on, given some evidence $x_{\boldsymbol{y}}$.

For the sake of clarity, let us suppose that the $\Phi(\boldsymbol{y}|x_{\boldsymbol{y}};\boldsymbol{w})$ can take two different shapes[8]:

[8]This decomposition into two layers will be a leitmotiv of the entire dissertation, since it highlights the double nature, symbolic and subsymbolic, of the reasoning process that can be carried on in MiniMax Entropy models in the relational setting

- On one hand we have $\Phi(\boldsymbol{y}|x_{\boldsymbol{y}};\boldsymbol{w}) = \frac{1}{N}\sum_c \Phi(\boldsymbol{y}^{(c)}|x_{\boldsymbol{y}}^{(c)};\boldsymbol{w})$. This class of potentials is the responsible of the conditioning of our distribution w.r.t. the feature representation x of constants. They should be responsible of telling how much some given truth assignments $\boldsymbol{y}^{(c)}$ to the ground atoms (symbolic) of a constant c are compatible with the feature representation of c, i.e. $x_{\boldsymbol{y}}^{(c)}$. For example, suppose constants c represent people and we have a feature representation of people ($g(c)$) telling us the age of individuals. Then, if we are reasoning about a constant `LittleJohn` whose age is 1 ($g(\texttt{LittleJohn}) = 1$), then we can promptly affirm that $p(\texttt{smokes(LittleJohn)=True})$ should be low, by a simple association between the feature representation of the constants and some of the ground truth fact about those constants. Without any claim of formality and generality, we can say that these potentials often concern single individuals and only few ground atoms of the current world, generally related to that individual only. This is related to what is generally referred to as subsymbolic reasoning or fast thinking (Kahneman, 2011)

- On the other hand we have $\Phi(\boldsymbol{y}|x_{\boldsymbol{y}};\boldsymbol{w}) = \Phi(\boldsymbol{y};\boldsymbol{w})$. This class of potentials are the responsible of the evaluation of the symbolic relational structure of data. They should be responsible of telling how much some given truth assignments $\boldsymbol{y}$ are compatible with each other given the underlying relational structure. For example, in the setting in which constants represent people, potentials should assign low probability to all those configuration of truth assignments to ground atoms where both `young(LittleJohn)` and `old(LittleJohn)` are assigned a `True` value. Here the reasoning process acts completely at the symbolic level without any need of recurring to the feature representation of constants. Clearly, the symbolic reasoning can take much more difficult forms. This is sometimes referred to as slow thinking (Kahneman, 2011).

These two class of potentials clearly show how the MiniMax entropy framework allows two very different reasoning process to take place. Obviously, while these two class of potential purposely encapsulate very distinct reasoning processes, in real settings, we expect potentials to take a behaviour that can be somewhere in the middle, where there is not anymore a clear distinction between subsymbolic/symbolic (fast/slow) reasoning.

3.4 Connection with classical settings

3.4.1 Supervised Learning

In a classical and pure supervised learning setup, the patterns x are i.i.d.. It is therefore possible to split the $\boldsymbol{y}$ into disjoint fragments γ_c of size $k = 1$, grouping the

atoms of a single constant c. Let us indicate as γ_c the fragment of the possible world restricted to a unique constant c, with $x_c = g(c)$.

A single parametrized conditional global potential Φ_S is needed to represent supervised learning, and this potential decomposes over correspondent fragment potentials as:

$$\Phi_S(\boldsymbol{y}|x;\boldsymbol{w}) = \sum_c \phi_S(\gamma_c, x_c;\boldsymbol{w})$$

Yielding,

$$p(\boldsymbol{y}|x,\boldsymbol{w},\beta) = \frac{1}{Z}\exp\left(\beta\sum_c \phi_S(\gamma_c, x_c;\boldsymbol{w})\right) \tag{3.19}$$

A possible implementation of the fragment potential ϕ_S is the dot product between the fragment γ_c and the output of a parametric model $f(x_c;w)$, for example a neural network. In this way, we are asking our MiniMax Entropy optimization problem to select the function f which mostly match the γ_c of our data, i.e.:

$$\phi_S(\gamma_c|x_c;\boldsymbol{w}) = \sum_i \gamma_{c_i} f_i(x_c;\boldsymbol{w}) \tag{3.20}$$

One-label classification. One-label classification is the task of assigning a single pattern to one and only one class over n possible classes. In the relational setting, we can say that we have c constants and n unary relations over them. Thus, requiring that only one unary relation can be *True* on a given constant c corresponds to a mutual exclusivity rule, which should assign a zero probability to worlds stating that a pattern can belong to more than one class.

Thus, we can add to the conditional potential above, representing supervised learning, another (non-conditional, non-parametrized) potential Φ_{ME} enforcing mutual-exclusivity, i.e.:

$$\Phi_{ME} = \sum_c \phi_{ME}(\gamma_c) \tag{3.21}$$

with:

$$\phi_{ME}(\gamma_c) = \begin{cases} -\infty & \text{if } \exists i,j : \gamma_{c,i} = \gamma_{c,j} = \texttt{True}; \\ 0 & \text{elsewhere} \end{cases} \tag{3.22}$$

Therefore, the corresponding model is:

$$p(\boldsymbol{y}|x,\boldsymbol{w},\beta) = \frac{1}{Z}\exp\left(\beta_S\sum_c \phi_S(\gamma_c, x_c;\boldsymbol{w}) + \beta_{ME}\sum_c \phi_{ME}(\gamma_c)\right) \tag{3.23}$$

where the partition function factorizes w.r.t. the single constants due to the independence of single constant in the supervised settings (i.e. the i.i.d. assumption).

$$Z = \sum_{\boldsymbol{y}' \in \mathcal{Y}} \exp\left(\beta_S \sum_c \phi_S(\gamma'_c, x_c; \boldsymbol{w}) + \beta_{ME} \sum_c \phi_{ME}(\gamma'_c)\right) \tag{3.24}$$

$$= \sum_{\boldsymbol{y}' \in \mathcal{Y}} \prod_c \exp\left(\beta_S \phi_S(\gamma'_c, x_c; \boldsymbol{w}) + \beta_{ME} \phi_{ME}(\gamma'_c)\right) \tag{3.25}$$

$$= \prod_c \sum_{\gamma'_c \in \Gamma} \exp\left(\beta_S \phi_S(\gamma'_c, x_c; \boldsymbol{w}) + \beta_{ME} \phi_{ME}(\gamma'_c)\right) \tag{3.26}$$

$$= \prod_c Z_c \tag{3.27}$$

The factorization of the partition function makes the entire model factorize w.r.t. the single constants:

$$p(\boldsymbol{y}|x, \boldsymbol{w}, \beta) = \prod_c \frac{1}{Z_c} \exp\left(\beta_S \phi_S(\gamma_c, x_c; \boldsymbol{w}) + \beta_{ME} \phi_{ME}(\gamma_c)\right) \tag{3.28}$$

$$= \prod_c p_c(\gamma_c|x_c; \boldsymbol{w}, \boldsymbol{\beta}) \tag{3.29}$$

Now, we can focus on two γ'_c in the partition functions Z_c:

- γ'_c where more than one atom is `True`: here, $\phi_{ME}(\gamma_c) = -\infty$ and thus they do not count in the computation of the partition function;
- γ'_c where only one atom i is `True`. Let us call $\gamma'_{c,i}$ the fragment γ'_c on the constant c with only one i-th ground atom set to `True`. These are the only fragments of the partition function Z_c that survive. It is easy to see that they are only n, where n was the number of unary predicates over constants. Thus:

$$Z_c = \sum_{i=1}^{n} \exp(\beta_S \phi_S(\gamma'_{c,i}, x_c; \boldsymbol{w})) \tag{3.30}$$

Finally, we obtained:

$$p_c(\gamma_c|x_c; \boldsymbol{w}, \boldsymbol{\beta}) = \begin{cases} 0 & \text{if } \exists i, j : \gamma_{c,i} = \gamma_{c,j} = \texttt{True}; \\ \dfrac{\exp\left(\beta_S \phi_S(\gamma_c, x_c; \boldsymbol{w})\right)}{\sum_{j=1}^{n} \exp(\beta_S \phi_S(\gamma'_{c,j}, x_c; \boldsymbol{w}))} & \text{only } \gamma_{c,i} = \texttt{True} \end{cases} \tag{3.31}$$

This is clearly a *softmax distribution* over the n relations defined on each constant c. The equality is even more evident if we substitute the functional form of ϕ_S:

$$p_c(\gamma_c|x_c;\boldsymbol{w},\boldsymbol{\beta}) = \begin{cases} 0 & \text{if } \exists i,j : \gamma_{c,i} = \gamma_{c,j} = \texttt{True}; \\ \dfrac{\exp\left(\beta_S f_i(x_c;\boldsymbol{w})\right)}{\sum_{j=1}^{n} \exp(\beta_S f_j(x_c;\boldsymbol{w}))} & \text{only } \gamma_{c,i} = \texttt{True} \end{cases} \tag{3.32}$$

which is the softmax activation function exploited on the linear outpur layer of a neural network f having x_c as inputs. This is arguably the most common setting of supervised learning in the modern deep learning scenario.

Multi-label. Suppose again we have multiple constants c and only unary relations. However, this time, we have only the supervised potential Φ_S and no other potentials creating dependence among the the different relations defined of a constant c. Here, all assignments to the fragment γ_c of a constant c are admissible, but they are completely independent from each other. We obtain a distribution which decomposes on all the constants and on all the relations.

By exploiting, $\phi_S(\gamma_c|x_c;\boldsymbol{w}) = \sum_i \gamma_{c_i} f_i(x_c;\boldsymbol{w})$, we have:

$$p(\boldsymbol{y}|x,\boldsymbol{w},\beta) = \frac{\exp\left(\sum_c \sum_{i=1}^{n} f_i(x_c;\boldsymbol{w}) \cdot \gamma_{c,i}\right)}{\sum_{\boldsymbol{y}'} \exp\left(\sum_c \sum_{i=1}^{n} f_i(x_c;\boldsymbol{w}) \cdot \gamma'_{c,i}\right)} = \tag{3.33}$$

$$= \prod_c \prod_{i=1}^{n} \frac{\exp\left(f_i(x_c;\boldsymbol{w}) \cdot \gamma_{c,i}\right)}{\sum_{\gamma'_{c,i} \in \{0,1\}} f_i(x_c;\boldsymbol{w}) \cdot \gamma'_{c,i}} \tag{3.34}$$

$$= \prod_c \prod_{i=1}^{n} \frac{\exp\left(f_i(x_c;\boldsymbol{w}) \cdot \gamma_{c,i}\right)}{1 + \exp\left(f_i(x)\right)} = \tag{3.35}$$

$$= \prod_{x \in S} \left[\prod_{i:\gamma_{c,i}=\texttt{True}} \sigma(f_i(x)) \cdot \prod_{i:\gamma_{c,i}=\texttt{False}} (1 - \sigma(f_i(x))) \right] \tag{3.36}$$

where $\sigma(\dot{})$ is the sigmoid or logistic function. This distribution is indeed a sigmoid distribution over both positive and negative examples. When learning with maximum likelihood, this scheme will recover exactly *logistic regression learning*.

3.4.2 Pure Logic reasoning

The mutual exclusivity potential $\Phi_{ME}(\boldsymbol{y})$, by acting only on the $\boldsymbol{y}$ variables, is a member of a larger class of potentials which extends the relational nature of the $\boldsymbol{y}$ variable and allows to describe symbolic features of data (see Section 3.3.2).

We can think the $\Phi_{ME}(\boldsymbol{y})$ as an n-ary exclusive or (XOR) over all the unary relation R_i of a constant c and thus thinking of the potential $\Phi_{ME}(\boldsymbol{y})$ as the one corresponding to a logical rule:

$$\forall c : R_1(c) \oplus R_2(c) \oplus \cdots \oplus R_n(c)$$

First Order Logic is a very powerful language to describe relational features of data and multiple rule-to-potential translation scheme could be designed. Chapters 4, 5 and 6 will share a specific scheme which allows FOL formulas to be translated into real-valued functions by means of fuzzy logic.

In this Section, we can show that a *counting* scheme of FOL formulas evaluation allows MiniMax Entropy model to recover exactly Markov Logic Networks (Richardson and Domingos, 2006) (see Section 2.2).

In particular, in a counting scheme, each FOL formula is considered universally quantified w.r.t. all its variables. Then, a potential can be designed to return, given an input possible world, the number of groundings which evaluates to `True` given the formula. In other worlds, let r be a FOL formula. Let γ be a fragment of $\boldsymbol{y}$ containing the groud atoms of a single grounding[9] of r. Finally let $\mathbb{1}_r(\gamma)$ be the indicator functions, which returns 1 if the formula evaluates to `True` for that particular grounding γ, returns 0 otherwise. Then, the potential $\Phi_r(\boldsymbol{y}) = \sum_{\gamma \in \Gamma_r} \mathbb{1}_r(\gamma)$.

Finally, given a collection R of rules, we can define the MiniMax Entropy model which exploit the counting scheme for FOL formulas as:

$$p(\boldsymbol{y}; \boldsymbol{\beta}) = \frac{1}{Z} \exp\Big(\sum_{r \in R} \beta_t \sum_{\gamma \in \Gamma} \mathbb{1}_r(\gamma)\Big) \tag{3.37}$$

which is exactly a Markov Logic Network.

3.5 Learning MiniMax Entropy models with MAP inference

3.5.1 Maximization as Expectation: a learning scheme for the conditional case

In Section 7.2.1, we have seen that the maximization of the log-likelihood of a MiniMax Entropy model requires the computation of the expected value of the potentials and of their gradients, whose exact computation is infeasible in most of the interesting cases.

[9] The close connection between groundings and fragments has been the one inspiring factorization over fragments.

We have also introduced (Section 3.2.7) some general methods to tackle the approximation of such expectations.

In this Section, we will see that MAP inference could provide us with a very simple (and strong) way of approximating expectations under the model. In the following Section, we will see how the same approximation can be interpreted in terms of variational inference.

The main argument of this approximation is that we assume that, given the feature representation of an unknown world (so it is restricted to the conditional case), the correspondent probability distribution will have a single predominant mode, i.e. the probability mass is mostly concentrated around a single maximum state. This is a meaningful hypothesis (Salakhutdinov and Hinton, 2009) for applications such as the interpretation of images or speech, where it can be also useful. Indeed, sacrificing some log-likelihood in order to make the true posterior unimodal could be advantageous for a system that must use the posterior to control its actions. Having many quite different and equally good representations of the same sensory input increases log-likelihood but makes it far more difficult to associate an appropriate action with that sensory input.

When this hypothesis is meaningful, then we can hypothesize that, for the sake of the evaluation of the expected value (so as an inference method only), the model probability distribution $p(\boldsymbol{y}|x;\theta)$ can be approximated by a δ of Dirac centred around the MAP state of the distribution p; i.e.:

$$p(\boldsymbol{y}|x;\theta) \approx \delta(\boldsymbol{y} - \boldsymbol{y}_{\text{MAP}})$$

where $\boldsymbol{y}_{\text{MAP}} = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|x;\theta)$ is the solution of the MAP inference.

It becomes immediately evident that this particular choice of the approximation turns the computation of expectations into a maximization problem aimed at finding the map state of the distribution. Indeed:

$$\begin{aligned}\mathbb{E}_p[f(\boldsymbol{y},x)] &= \sum_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{y},x)p(\boldsymbol{y}|x) \\ &= \sum_{\boldsymbol{y}\in\mathcal{Y}} f(\boldsymbol{y},x)\delta(\boldsymbol{y} - \boldsymbol{y}_{\text{MAP}}) \\ &= f(\boldsymbol{y}_{\text{MAP}},x)\end{aligned}$$

Using this approximation, we can rewrite the gradients of Section 7.2.1 as:

$$\frac{\partial \log(p(\widehat{\boldsymbol{y}}|x))}{\partial w_{i,j}} = \beta_i\left(\frac{\partial \Phi_i(\widehat{\boldsymbol{y}}|x;\boldsymbol{w}_i)}{\partial w_{i,j}} - \frac{\partial \Phi(\boldsymbol{y}_{\text{MAP}}|x;\boldsymbol{w}_i)}{\partial w_{i,j}}\right) \tag{3.38}$$

$$\frac{\partial \log(p(\widehat{\boldsymbol{y}}|x))}{\partial \beta_i} = \Phi_i(\widehat{\boldsymbol{y}}|x;\boldsymbol{w}_i) - \Phi_i(\boldsymbol{y}_{\text{MAP}}|x;\boldsymbol{w}_i) \tag{3.39}$$

We can now devise a new learning scheme which recalls the one in Algorithm 1, but substitutes the expectations step, with a maximization step.

To this end, it is interesting to underline an interpretation of MAP inference as approximate inference, as we will describe in the following Section.

3.5.2 MAP inference as approximated inference

MAP inference is usually not thought of as approximate inference - it does compute the exact most likely value of $\boldsymbol{y}^\star$. However, following variational inference principle, if we want to develop a learning process based on maximizing the variational lower-bound $\mathcal{L}(q)$ then it would be helpful to think of MAP inference as a procedure that provides a value of q. In this sense, we can think of MAP inference as approximate inference, because it does not provide the optimal q.

From Section 3.2.7, exact inference consists of maximizing

$$\log p(x;\theta) = \mathcal{L}(q) + KL(q||p)$$

with respect to q over an unrestricted family of probability distributions, using an exact optimization algorithm. Since the left-hand side of this equation is constant w.r.t. q, maximizing $\mathcal{L}(q)$ means minimizing the KL divergence between q and $p(\boldsymbol{y}, x;\theta))$. We can derive MAP inference as a form of approximate inference by restricting the family of distributions q may be drawn from. Specifically, in the setting of MAP inference as approximated inference, we require q to take on a Dirac distribution, as done in the previous Section:

$$q(\boldsymbol{y}) = \delta(\boldsymbol{y} - \mu)$$

This means that we can now control q entirely via μ. Dropping terms of $\mathcal{L}$ that do not vary with μ, we are left with the optimization problem

$$\boldsymbol{y}^\star = \arg\max_{\boldsymbol{y}} p(\boldsymbol{y}|x;\theta)$$

which is equivalent to the MAP inference problem.

We can thus justify a learning procedure in which we alternate between performing MAP inference to infer $\boldsymbol{y}^\star$ and then update θ to increase $\log p(\boldsymbol{y}^\star|v)$. This recovers a different interpretation of Expectation Maximization algorithm, given by Neal and Hinton (1998). Here, the expectation step is interpreted as the process aimed at finding a $q(\boldsymbol{y}|x) = p(\boldsymbol{y}|x;\hat{\theta})$, where $\hat{\theta}$ is the current estimation of the posterior parameters. Therefore, the E step can be carried out by variational inference procedures.

Thus, the maximization of the likelihood of our MiniMax Entropy model has automatically pointed to an algorithm for its optimization. This algorithm retraces a standard EM algorithm for graphical model learning. When coupled with varational inference methods, this algorithm recovers another interpretation of the E step, in terms of inferring a proposal distribution for the current estimated posterior.

No matter which interpretation we are going to give to the algorithm, this scheme captures the intrinsic nature of the learning process (Goodfellow et al., 2016). Indeed, we update the model parameters to improve the likelihood of a complete dataset, where all missing variables have their values provided by an estimate of the posterior distribution. This particular insight is not unique to the EM algorithm. For example, using gradient descent to maximize the log-likelihood also has this same property; the log-likelihood gradient computations require taking expectations with respect to the posterior distribution over the hidden units.

3.5.3 MAP approximation with functional approximation

We can think of the MAP inference in Section 3.3.1 as a function $M(x)$ that, given the feature representation of constants x, provides the MAP state under the distribution; i.e.:

$$\boldsymbol{y}_{\text{MAP}} = M(x) \tag{3.40}$$

Now, let us suppose that this function M can be approximated by a universal approximator f (e.g. a neural network), giving:

$$\boldsymbol{y}_{\text{MAP}} = M(x) \approx f(x; \boldsymbol{q}) \tag{3.41}$$

with $\boldsymbol{q}$ the set of parameters of the approximator f.

Gradients of Section 7.2.1 now become:

$$\frac{\partial \log(p(\widehat{\boldsymbol{y}}|x))}{\partial w_{i,j}} = \beta_i \left(\frac{\partial \Phi_i(\widehat{\boldsymbol{y}}|x; \boldsymbol{w}_i)}{\partial w_{i,j}} - \frac{\partial \Phi_i(f(x; \boldsymbol{q})|x; \boldsymbol{w}_i)}{\partial w_{i,j}} \right) \tag{3.42}$$

$$\frac{\partial \log(p(\widehat{\boldsymbol{y}}|x))}{\partial \beta_i} = \Phi_i(\widehat{\boldsymbol{y}}|x; \boldsymbol{w}_i) - \Phi_i(f(x; \boldsymbol{q})|x; \boldsymbol{w}_i) \tag{3.43}$$

Since the f should approximate a MAP inference step, its training can be described in terms of the following optimization problem:

$$\max_{\boldsymbol{q}} \sum_i \beta_i \Phi_i(f(x; \boldsymbol{q})|x; \boldsymbol{w}_i)$$

3.5.4 Learning from constraints

In this Section, we aim at showing that the framework of learning from constraints (LfC) (Gori, 2017) can be interpreted as a very particular instantiation of a MiniMax Entropy model, even if starting from a completely different viewpoint.

Indeed, the framework of learning from constraints does not make any assumption about the existence of a probabilistic distribution over possible worlds. The framework is instead based on regularization theory, classically exploited in the SVM literature. The main ingredients are:

- The learning process aims at finding good approximators of the predicate functions under a regularization principle. Ground atoms are then seen as predictions of some functions f working on the feature representation of constants.

- Potentials are considered as constraints of the world, describing some known behaviour (i.e. prior knowledge) of the data. Therefore, they are always considered known and valid everywhere, and then any potential evaluated on the data $\Phi(\widehat{\boldsymbol{y}})$ is equal to its maximum value. As a direct consequence, the role of the β parameters should be considered as a relative weight between different Φ_i to guide the learning process, and not as a *likelihood* degree, otherwise they will all take a very large value. In this framework, they are treated as hyperparameters.

We will go into much more details in Chapter 4. However, it is useful in this context to examine the shape of the objective function of the LfC framework, in order to better analyse the links with the MiniMax Entropy framework.

In the LfC framework, one builds a loss function of the following shape:

$$L(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w}) = \sum_i Cost(f(x_i; \boldsymbol{w}), y_i) + \lambda r(\boldsymbol{w}) + \sum_k \beta_k \Psi_k(f(\boldsymbol{x})),$$

where *Cost* is a classical cost function penalizing truth degree predicted by the f functions far from the ground truth, $\dot{r}()$ is a regularization term penalizing functions which are too complex, and Ψ_k are constraints[10] specifying a specific expected behaviour of the functions (i.e. of the ground atoms) and penalizing behaviours different from the expected one.

However, even if very different in nature, there are similarities that can be shown, linking the LfC framework and the MiniMax Entropy framework.

By asking some functions to approximate the truth degree of some atoms, the LfC framework is implicitely assuming that, given some feature representation, there is a unique predominant explanation. This is indeed exactly what the MAP

[10] They have an inverse role of potentials and, thus, they can be interpreted in terms of energies.

approximation is doing. Thus, the predicate approximators f of the LfC framework can be seen as the MAP approximators of Section 3.5.3.

Suppose we design a MiniMax Entropy model where we design a specific potential $\Phi_r(\boldsymbol{y}|x;\boldsymbol{w}) = -Dist(\boldsymbol{y}, h(x;\boldsymbol{w}))$. The idea is that our model wants to consider more likely those worlds that can be inferred simply by the constants feature representations by means of a non-linear function h. Moreover, let us assume that there are also other known potentials defined only on the symbolic domain and expressed as negative cost function; i.e. $\Phi_k(\boldsymbol{y}) = -\Psi_k(\boldsymbol{y})$.

The corresponding model is:

$$\begin{aligned} p(\boldsymbol{y}|x;\boldsymbol{w}) &= \frac{1}{Z}\exp(\Phi_r(\boldsymbol{y}|x;\boldsymbol{w}) + \sum_k \beta_k \Phi_k(\boldsymbol{y})) \\ &= \frac{1}{Z}\exp\Big(-\big(Dist(\boldsymbol{y}, h(x;\boldsymbol{w})) + \sum_k \beta_k \Psi_k(\boldsymbol{y})\big)\Big) \end{aligned}$$

Then, under the functional approximation of the MAP inference, the gradients in Equation 3.43 becomes:

$$\begin{aligned} \frac{\partial \log(p(\widehat{\boldsymbol{y}}|x))}{\partial w_{i,j}} &= \beta_i\left(\frac{\partial \Phi_i(\widehat{\boldsymbol{y}}|x;\boldsymbol{w}_i)}{\partial w_{i,j}} - \frac{\partial \Phi_i(f(x;\boldsymbol{q})|x;\boldsymbol{w}_i)}{\partial w_{i,j}}\right) \\ &= \frac{\partial}{\partial w_{i,j}}\Big(Dist(f(x;q), h(x;\boldsymbol{w})) - Dist(\widehat{\boldsymbol{y}}, h(x;\boldsymbol{w}))\Big) \end{aligned}$$

Since this gradient are exploited in a gradient step for maximizing the log-likelihood, we can identify an equivalent objective:

$$\min_{\boldsymbol{w}} \Big[Dist(\widehat{\boldsymbol{y}}, h(x;\boldsymbol{w})) - Dist(f(x;q), h(x;\boldsymbol{w}))\Big]$$

On the other end, as shown previously, the $f(x;q)$ functions are trained to converge to the MAP state, which is equivalent to minimizing the negative exponent of the MiniMax Entropy distribution:

$$\min_{\boldsymbol{q}} \big[Dist(f(x;q), h(x;\boldsymbol{w})) + \sum_k \beta_k \Psi_k(f(x;\boldsymbol{q}))\big]$$

Now, if we look carefully, we have here two function approximators which accomplish two different tasks.

First, the function $h(x;\boldsymbol{w})$ is shaping the potential Φ_r in order to maximize the likelihood of the training data. It is the only responsible for maximizing the likelihood

since the other potentials have no parameters (the β here is considered a constant for the model). Indeed, they provide a probability mass which is independent of the likelihood. No matter what the data shows, the states, which satisfy the constraints, get always an higher probability than those who do not satisfy them.

Second, the function $f(x;\boldsymbol{q})$ is approximating a MAP inference step, thus it tries to predict states which are close to the $h(x;q)$ function (close to the data and thus maximizing the likelihood) but at the same time satisfying the constraints.

What one could do, since f are asked to match h, is to impose $f(x;\boldsymbol{q}) = h(x;\boldsymbol{w})$. Then, we can merge the two optimization problems above into a unique one, which concerns only the set $\boldsymbol{q}$ of parameters and where all the terms bringing f and h close to each other clearly cancel.

$$\min_{q} Cost(\widehat{\boldsymbol{y}}, f(x;\boldsymbol{q})) + \sum_{k} \beta_k \Psi_k(f(x;\boldsymbol{q})) \tag{3.44}$$

Interestingly, this is exactly the objective function of the LfC framework, apart from the regularization term that clearly comes from a prior $p(\boldsymbol{q})$ on the parameters of the functions f. Indeed, it can be easily recovered by moving MiniMax Entropy models to a more Bayesian approach where we provide priors to all the parameters. Weight decay is usually recovered by applying a gaussian prior on q (see Chapter 6).

3.6 Discussion

As a final discussion, it is worth noticing that the MiniMax Entropy models introduced in this chapter are an extremely large class of learning models that, as shown in Section 3.5.4, can incorporate also very different theories. The main reason is that the optimization problem merging *learning* (e.g. minimization of entropy) and *generalization* (e.g. maximization of entropy) is likely to be a fundamental ingredient of all intelligent behaviours.

It is clear that there exist multiple ways a MiniMax Entropy model can be exploited in an effective learning algorithm. In this chapter, we want to focus to two design choices usually fundamental in instantiating the MiniMax Entropy theory into an effective algorithm, i.e. *potential design* and *inference algorithm*. These design choices require sometime *assumptions* and sometime *approximations* of the problem under investigation.

Potential Design In the rest of the chapter, it has been shown that potentials are a fundamental ingredient of the overall theory. When *known* (*given*), they correspond to the words of a language we provide our MaxEnt model to describe the data we are observing. When *unknown* (*to be learned*), we are asking our MinEnt model to

find those words which are more expressive of the phenomenon under investigation. Therefore, a clear distinction must be done between *given* and *trainable* potentials.

Another design choice regarding potentials is their *factorization* properties. This choice is deeply connected to the choice of the inference algorithm. Most of the methods for making inference in probabilistic graphical models scalable rely on some assumptions about the factorization properties of potentials and, thus, of the partition function.

Finally, potentials differ for their *differentiability* properties. We have seen that some inference algorithm (e.g. variational inference) are based on the possibility of translating the inference problem into an optimization problem. Thus, differentiable potentials allow the exploitation of a wider class of inference methods.

Inference Algorithm If given with infinite time, a naive computation of the expectations in Equations 3.17 and 3.16 would provide a perfect learning algorithm for the maximization of the likelihood on some given data. However, apart for very trivial problem, an exact computation of those expectations is not feasible with the modern hardware and we must rely on smarter or approximate ways of computing the expectations. Thus, the learning capability of our MiniMax Entropy model by a specific algorithm strongly depend on the choice of the inference methods.

In the following chapters, we will see how different (but correlated) choices of the potentials and the inference algorithm give raise to different algorithms for learning in the relational setting. In particular, Chapters from 4 to 6 show as we can exploit conditional MiniMax entropy models and known potentials derived from logical arguments to augment the possibilities of standard neural learners. Chapter 7 shows how we can exploit the representative power of neural networks to learn complex potentials on the relational setting.

Chapter 4

T-Norm Fuzzy Logic Potentials

In this Chapter, we introduce a novel method to define potentials based on prior knowledge on the problem expressed using logic formalism. The use of fuzzy logics allows us to obtain differentiable potentials, which enables the use of MAP inference as inference method. As showed in Section 3.5.4, this particular instantiation of the MiniMax Entropy setting recovers the learning from constraints LfC (Gori, 2017) setting, where potentials with fixed weights can be considered constraints on the learning process. In this Chapter, we exploit this last interpretation since it is has been the inspiring one. In Section 4.1, we will first describe the Learning From Constraint framework under the original interpretation of parsimonious learning. Then, in Section 4.2 we introduce the fundamentals of t-norm fuzzy logic, the underlying mathematical theory allowing the translation of logical formulas into real valued functions. Then, we show how t-norm formulas can be exploited to design differentiable constraints and how generators of Archimedean t-norms turn out to be a natural ingredient of this design. Finally, we show how a logic theory can be exploited to constrain a learning problem under the LfC framework.

4.1 Learning From Constraints

In this section, we recall a formulation of learning from constraints (Gnecco et al., 2015; Gori, 2017) based on a variational formulation of the parsimony principle, which aims at keeping small a functional that involves the function to be learned and its derivatives up to some order via suitable linear differential operators.

In this setting, the presence of a perceptual space $X \subset \mathbb{R}^d$ is assumed. The goal of the learning is interpreted as learning the behaviour on an intelligent agent implementing a vectorial function $f := [f_1, ..., f_n] \in \mathcal{F}$,where $\mathcal{F}$ is a space of functions from X to $\mathbb{R}^n$. The function $f_j, j = 1, \ldots, n$ is called the j-th task of the agent and $\mathcal{F}$ the task space.

The link with the relational setting in Section 3.1 is recovered once one thinks

at the task functions f_j as a functional implementation of the predicates of a FOL language, thus $f_j : X \to \{0,1\}$. By relaxing the logic to the fuzzy case, as will be the case of this Chapter, $f_j : X \to [0,1]$.

The interaction between the agent and the environment is modeled by constraints that have to be strictly satisfied *(hard constraints*) or constraints that can be violated, at the cost of some penalization quantified by a loss (*soft constraints*).

In order for the theory for being well defined, one has to assure that the task space $\mathcal{F}$ belongs to a specific Sobolev space. In particular, $\forall j \in N_n := \{1, ..., n\}$ and some positive integer k, the function $f_j : X \to \mathbb{R}$ belongs to the Sobolev space $\mathcal{W}^{k,2}(X)$, that is, the subset of $\mathcal{L}^2(X)$ whose elements f_j have weak partial derivatives up to the order k with finite $\mathcal{L}^2(X)$- norms.

To define the learning model, a parsimony index is introduced, defined via a differential operator. We consider linear-differential operators that are invariant under spatial shift and have constant coefficients. In particular, the vectorial finite-order linear differential operator $P := [P_0, \ldots, P_{l-1}]$ as the l-tuple of operators P_i , $i = 0, \ldots, l-1$, acting on the Sobolev space $\mathcal{W}^{k,2}(X)$ and such that:

$$P_i := \sum_{|\alpha| \leq k_i} b_{i,\alpha} D^\alpha$$

where α is a multi-index with n nonnegative components α_j, $D^\alpha f = \frac{\partial^{|\alpha|}}{\partial \alpha_1 \ldots \partial \alpha_n} f$.

Let P be a finite-order linear differential operator. Let be $||f_j||_P^2$ be a norm under this differential operator; i.e. $||f_j||_P^2 := \langle Pf_j, Pf_j \rangle$. Let $\gamma \in \mathbb{R}^n$ a vector of positive components.

Definition 4.1 (Parsimony Index)**.** *The parsimony index over a vectorial function $f = [f_1, ..., f_n] \in \mathcal{F}$ is defined as:*

$$\mathcal{E} := ||f||_{P,\gamma} = \sum_{j=1}^{n} \gamma_j ||f_j||_P^2$$

Thus, the general principle for learning the behaviour of a *parsimonious agent* can be formulated

Definition 4.2 (Parsimonious agent)**.** *A parsimonious agent that interacts with the environment aims at minimizing over the task space the functional $\mathcal{E}$. More specifically, in the case of hard constraints, the functional $\mathcal{E}$ has to be minimized on the subset of $\mathcal{F}$ of the task space that satisfies the hard constraints, whereas in the soft case, one has to minimize on the entire $\mathcal{F}$ the sum of equation $\mathcal{E}$ and a suitable penalty term, which quantifies the violation of the given set of soft constraints. In the case of hard constraints mixed with soft ones, the sum above is minimized on the subset of F of the task space that satisfies the hard constraints.*

In particular, let $\psi_c(f) : \mathcal{F} \to \mathbb{R}$ be a penalty term which penalizes violation of the given set of functions f with respect to a given constraint c. Let $\mathcal{C} = \mathcal{C}_h \cup \mathcal{C}_s$ be a collection of hard (h) and soft (s) constraints, whose definition is not important if, for each $c \in \mathcal{C}$, $\psi_c(f)$ is unambiguously defined. Then, the learning problem can be stated as:

$$\begin{aligned} \min_f \quad & \mathcal{E}(f) + \sum_{c \in \mathcal{C}_s} \lambda_c \psi_c(f) \\ \text{s.t.} \quad & \psi_c(f) = 0, \forall c \in \mathcal{C}_h \end{aligned} \tag{4.1}$$

with $\lambda = \{\lambda_c\}$ a set of suitable real coefficients, referred to as *beliefs* of the soft constraints.

4.2 Fundamentals of T-Norm Fuzzy Logic

Many-valued logics have been introduced in order to extend the admissible set of truth values from *true* (1), *false* (0) to a scale of truth-degree having *absolutely true* and *absolutely false* as boundary cases. In particular, in fuzzy logic the set of truth values coincides with the real unit interval $[0, 1]$. In this section, the basic notions of fuzzy logic together with some remarkable examples are introduced. According to (Hájek, 2013), a fuzzy logic can be defined upon a certain *t-norm* (triangular norm) representing an extension of the Boolean conjunction.

Definition 4.3 (t-norm)**.** *$T : [0, 1]^2 \to [0, 1]$ is a t-norm if and only if for every $x, y, z \in [0, 1]$:*

$$\begin{gathered} T(x, y) = T(y, x), \quad T(x, (T(y, z)) = T(T(x, y), z), \\ T(x, 1) = x, \qquad\qquad T(x, 0) = 0, \\ x \leq y \quad \longrightarrow \quad T(x, z) \leq T(y, z)\ . \end{gathered}$$

T is a continuous t-norm if it is continuous as function.

Table 4.1 reports the algebraic definition of t-norms and other logical operators definable by the chosen t-norm for Gödel, Łukasiewicz and Product logics respectively, which are referred as the fundamental fuzzy logics because all the continuous t-norms can be obtained by ordinal sums (Mostert and Shields, 1957; Jenei, 2002). The notation of the logical operators in Table 4.1 is given by the following definitions according to a certain t-norm T:

	Gödel	Łukasiewicz	Product
$x \otimes y$	$\min\{x, y\}$	$\max\{0, x + y - 1\}$	$x \cdot y$
$x \Rightarrow y$	$x \leq y ? 1 : y$	$\min\{1, 1 - x + y\}$	$x \leq y ? 1 : \frac{y}{x}$
$x \Leftrightarrow y$	$x \leq y ? x : y$	$1 - \lvert x - y \rvert$	$x = y ? 1 : \min\{\frac{x}{y}, \frac{y}{x}\}$
$x \wedge y$	$\min\{x, y\}$	$\min\{x, y\}$	$\min\{x, y\}$
$x \vee y$	$\max\{x, y\}$	$\max\{x, y\}$	$\max\{x, y\}$
$\sim x$	$x = 0 ? 1 : 0$	$1 - x$	$x = 0 ? 1 : 0$
$\neg x$	$1 - x$	$1 - x$	$1 - x$
$x \oplus y$	$\max\{x, y\}$	$\min\{1, x + y\}$	$x + y - x \cdot y$
$x \to y$	$\max\{1 - x, y\}$	$\min\{1, 1 - x + y\}$	$1 - x + x \cdot y$

Table 4.1: The truth functions for the residuum, bi-residuum, weak conjunction, weak disjunction, residual negation, strong neation, t-conorms and material implication of the fundamental fuzzy logics.

Definition 4.4 (definable connectives from a t-norm)**.**

$$
\begin{aligned}
&(\textit{t-norm}) && x \otimes y = T(x, y)\\
&(\textit{residual-impl}) && x \Rightarrow y = \max\{z : x \otimes z \leq y\}\\
&(\textit{bi-residuum}) && x \Leftrightarrow y = (x \Rightarrow y) \otimes (y \Rightarrow x)\\
&(\textit{weak-conj}) && x \wedge y = x \otimes (x \Rightarrow y)\\
&(\textit{weak-disj}) && x \vee y = ((x \Rightarrow y) \Rightarrow y) \otimes ((y \Rightarrow x) \Rightarrow x)\\
&(\textit{residual-neg}) && \sim x = x \Rightarrow 0\\
&(\textit{strong-neg}) && \neg x = 1 - x\\
&(\textit{t-conorm}) && x \oplus y = \neg(\neg x \otimes \neg y)\\
&(\textit{material-impl}) && x \to y = \neg x \oplus y
\end{aligned}
$$

4.2.1 Archimedean T-Norms

In mathematics, t-norms (Klement et al., 2004a, 2013) are a special kind of binary operations on the real unit interval $[0, 1]$ especially used in engineering applications of fuzzy logic. Table 4.1 reports the fundamental continuous t-norms, 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 by rotations or ordinal sums of other t-norms or defining parametric classes. This section introduces *Archimedean* t-norms (Klement et al., 2004c), a special class of t-norms that can be constructed by means of unary monotone functions, called *generators*.

Definition 4.5. *A t-norm T is said to be* Archimedean *if for every $x \in (0, 1)$, $T(x, x) < x$. In addition, T is said* strict *if for all $x \in (0, 1)$, $0 < T(x, x) < x$ otherwise is said* nilpotent.

For instance, the Łukasiewicz (T_L) and Product (T_P) t-norms are respectively nilpotent and strict, while the Gödel (T_G) t-norm is not archimedean, indeed it is *idempotent* ($T_G(x,x) = x$, for all $x \in [0,1]$). In addition, Łukasiewicz and Product t-norms are enough to represent the whole classes of nilpotent and strict Archimedean t-norms (Klement et al., 2013).

Theorem 3. *Any nilpotent t-norm is isomorphic to T_L and any strict t-norm is isomorphic to T_P.*

A fundamental result for the construction of t-norms by *additive generators* is based on the following theorem (Klement et al., 2004b).

Theorem 4. *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 $[0,1]$, and $g^{(-1)}$ its pseudo-inverse. Then the function $T : [0,1] \to [0,1]$ defined as*

$$T(x,y) = g^{-1}\left(\min\{g(0^+), g(x) + g(y)\}\right) \ . \tag{4.2}$$

is a t-norm and g is said an additive generator *for T. T is strict if $g(0^+) = +\infty$, otherwise T is nilpotent. .*

Example 4.1. *If we take $g(x) = 1 - x$, we get T_L*

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

while taking $g(x) = -\log(x)$, we get T_P

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

An interesting consequence of equation (4.2) is that it allows us to define also the other fuzzy connectives, deriving from the t-norm, as depending on the additive generator. For instance:

$$\begin{aligned} x \Rightarrow y &= g^{-1}\left(\max\{0, g(y) - g(x)\}\right) \\ x \Leftrightarrow y &= g^{-1}\left(|g(x) - g(y)|\right) \\ x \oplus y &= 1 - g^{-1}\left(\min\{g(0^+), g(1-x) + g(1-y)\}\right) \end{aligned} \tag{4.3}$$

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_L and T_P, respectively. Additive and multiplicative generators are isomorphic and we decide to focus on the former for simplicity. We only mention that both multiples of additive generators and positive powers of multiplicative generators determine the same t-norm.

4.2.2 Parameterized classes of t-norms

Given a generator of a t-norm depending on a certain parameter, we can define a class of related t-norms depending on such parameter. For instance, given a 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. In addition, let T_D and T_G denote the *Drastic* (defined by $T_D(x,y) = (x = y = 1)?1 : 0$) and Gödel t-norms respectively, we get:

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

On the other hand, several parameterized families of t-norms have been introduced and studied in the literature (Klement et al., 2013). In the following we recall some prominent examples we will exploit in the experimental evaluation.

Definition 4.6 (The Schweizer-Sklar family)**.** *For $\lambda \in (-\infty, +\infty)$, consider:*

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

The t-norms corresponding to this generator are called Schweizer-Sklar t-norms, and they are defined according to:

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

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 $\lambda \in [-\infty, +\infty]$, in addition $T_1^{SS} = T_L$.*

Definition 4.7 (Frank t-norms). *For $\lambda \in [0, +\infty]$, consider:*

$$g_\lambda^F(x) = \begin{cases} -\log(x) & \text{if } \lambda = 1 \\ 1 - x & \text{if } \lambda = +\infty \\ \log(\frac{\lambda - 1}{\lambda^x - 1}) & \text{otherwise.} \end{cases}$$

The t-norms corresponding to this generator are called Frank t-norms and they are strict if $\lambda < +\infty$. The overall class of Frank t-norms is decreasing and continuous.

$$T_\lambda^F(x, y) = \begin{cases} T_G & \text{if } \lambda = 0 \\ T_P & \text{if } \lambda = 1 \\ T_L & \text{if } \lambda = +\infty \\ \log_\lambda \left(1 + \frac{(\lambda^x - 1)(\lambda^y - 1)}{\lambda - 1}\right) & \text{otherwise.} \end{cases}$$

4.2.3 Loss Functions by T-Norms Generators

In this section, we present a novel approach to combine the choice of both the fuzzy conversion of formulas and the penalty map according to a unified principle. In particular, we investigate the mapping of formulas into constraints by means of generated t-norm fuzzy logics, and we exploit the same additive generator of the t-norm to map the formulas to be satisfied into the functional constraints to be minimized, i.e. we consider $L = g$. Moreover, since the quantifiers can be seen as generalized AND and OR over the grounded expressions (see Remark 1), we show that the same fuzzy conversion, so as the overall loss function, as expressed in Equation 4.6, only depends on the chosen t-norm generator.

Remark 1. *Given a formula $\varphi(x)$ defined on $\mathcal{X}$, the role of the quantifiers have to be interpreted as follows,*

$$\begin{aligned} \forall x\, \varphi(x) &\simeq \varphi(x_1)\ AND\ \ldots\ AND\ \varphi(x_N) \\ \exists x\, \varphi(x) &\simeq \varphi(x_1)\ OR\ \ldots\ OR\ \varphi(x_N) \end{aligned}$$

where $\mathcal{X} = \{x_1, \ldots, x_N\}$ denotes the available samples.

Given a certain formula $\varphi(x)$ depending on a variable x that ranges in the set $\mathcal{X}$ and its corresponding functional representation $f_\varphi(x, \boldsymbol{p})$, the conversion of any universal quantifier may be carried out by means of an Archimedean t-norm T, while the existential quantifier by a t-conorm. For instance, given the formula $\psi = \forall x\, \varphi(x)$, we have:

$$f_\psi(\mathcal{X}, \boldsymbol{p}) = g^{-1}\left(\min\left\{g(0^+), \sum_{x \in \mathcal{X}} g\big(f_\varphi(x, \boldsymbol{p})\big)\right\}\right) \tag{4.4}$$

where g is an additive generator of the t-norm T.

Since any generator function is decreasing and $g(1) = 0$, the generator function is a very natural choice to be used as loss L which can be used to map the fuzzy conversion of the formula, as reported in Equation 4.4, in a constraint to be minimized. By exploiting the same generator of T to map into a loss function, we get the following term $L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big)$ to be minimized:

$$L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) = \begin{cases} \min\left\{ g(0), \sum_{x \in \mathcal{X}} g(f_\varphi(x, \boldsymbol{p})) \right\} & T \text{ nilpotent} \\ \sum_{x \in \mathcal{X}} g(f_\varphi(x, \boldsymbol{p})) & T \text{ strict} \end{cases} \tag{4.5}$$

As a consequence, the following result can be provided with respect to the convexity of the loss $L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big)$.

Proposition 1. *If g is a linear function and f_φ is concave, $L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big)$ is convex. If g is a convex function and f_φ is linear, $L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big)$ is convex.*

Proof. Both the arguments follow since if f_φ is concave (we recall that a linear function is both concave and convex, as well) and g is a convex non-increasing function defined over a univariate domain, then $g \circ f_\varphi$ is convex. □

Proposition 1 establishes a general criterion to define convex constraints according to a certain generator depending on the fuzzy conversion f_φ and, in turn, by the logical expression φ. In Example 4.2 are reported some application cases.

$$\underbrace{g^{-1}\left(\sum_x g\left(\underbrace{g^{-1}\left(\max\left\{0, g(p_3(x)) - g\left(\underbrace{g^{-1}(g(p_1(x)) + g(p_2(x)))}_{conjunction}\right)\right\}\right)}_{implication}\right)\right)}_{quantifier}$$
$$= g^{-1}\left(\sum_x \max\left\{0, g(p_3(x)) - g(p_1(x)) - g(p_2(x))\right\}\right)$$

Table 4.2: Example of the translation of $\forall x \; p_1(x) \otimes p_2(x) \Rightarrow p_3(x)$ with respect to the selection of a t-norm generator g. The simplification expressed on the right side is general and can be applied for a wide range of logical operators.

Example 4.2. *If $g(x) = 1 - x$ we get the Łukasiewicz t-norm, that is nilpotent. Hence, from Equation 4.5 we get:*

$$L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) = \min(1, \sum_{x \in \mathcal{X}} (1 - (f_\varphi(x, \boldsymbol{p}))) \, .$$

In case f_φ is concave (Giannini et al., 2018), this function is convex.

If $g(x) = -\log(x)$ (Product t-norm) from Equation 4.5 we get a generalization of the cross-entropy loss:

$$L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) = -\sum_{x \in \mathcal{X}} \log(f_\varphi(x)) \,.$$

In case $f_\varphi(x)$ is linear (e.g. a literal), this function is convex.

So far, we only considered the case of a general formula φ. In the following, different cases of interest for φ are reported. Given an additive generator g for a t-norm T, additional connectives may be expressed with respect to T, as reported e.g. by Equation 4.3. If p_1, p_2 are two unary predicate functions sharing the same input domain $\mathcal{X}$, the following formulas yield the following penalty terms, where we supposed T strict for simplicity:

$$\begin{aligned} \forall x\, p_1(x) &\longrightarrow \sum_{x \in \mathcal{X}} g(p_1(x)) \\ \forall x\, p_1(x) \Rightarrow p_2(x) &\longrightarrow \sum_{x \in \mathcal{X}} \max\{0, g(p_2(x)) - g(p_1(x))\} \\ \forall x\, p_1(x) \Leftrightarrow p_2(x) &\longrightarrow \sum_{x \in \mathcal{X}} |g(p_1(x)) - g(p_2(x))| \end{aligned}$$

According to a certain generator, different loss functions may arise from the same FOL formula. 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 for the cross-entropy loss (see Example 4.2).

Example 4.3. *If $g(x) = \frac{1}{x} - 1$, with corresponding strict t-norm $T(x, y) = \frac{xy}{x+y-xy}$, the functional constraint 4.5 that is obtained applying g to the formula $\forall x\, p_1(x) \Rightarrow p_2(x)$ is given by*

$$L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) = \sum_{x \in \mathcal{X}} \max\left\{0, \frac{1}{p_2(x)} - \frac{1}{p_1(x)}\right\} \,.$$

While if $g(x) = 1 - x^2$, with corresponding nilpotent t-norm $T(x, y) = \min\{1, 2 - x^2 - y^2\}$, the constraint is given by

$$L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) = \min\left\{1, \sum_{x \in \mathcal{X}} \max\left\{0, (p_1(x))^2 - (p_2(x))^2\right\}\right\} \,.$$

4.2.4 The simplification property

An interesting property of this method consists in the fact that, in case of compound formulas, some occurrences of the generator may be simplified. For instance, this is shown in Table 4.2 for the formula $\forall x\, p_1(x) \otimes p_2(x) \Rightarrow p_3(x)$. However, this property

does not hold for all the connectives that are definable upon a certain generated t-norm (see Definition 4.4). For instance, $\forall x\ p_1(x) \oplus p_2(x)$ becomes:

$$g\Big(1 - g^{-1}(\min\{g(0^+), \sum_x g(1 - p_1(x)) + g(1 - p_2(x))\})\Big)$$

This suggests to identify the connectives that, on one hand allow the simplification of any occurrence of g^{-1} by applying g in its corresponding functional expression, and on the other hand allow the evaluation of g only on grounded predicates. For short, in the following we say that the formulas build upon such connectives have the *simplification property*.

Lemma 1. *Any formula φ, whose connectives are restricted to $\{\wedge, \vee, \otimes, \Rightarrow, \sim, \Leftrightarrow\}$, has the simplification property.*

Proof. The proof is by induction with respect to the number $l \geq 0$ of connectives occurring in φ.

- If $l = 0$, i.e. $\varphi = p_j(x_i)$ for a certain $j \leq J$, $x_i \in \mathcal{X}$; then $g(f_\varphi) = g(p_j(x_i))$, hence φ has the simplification property.
- If $l = k + 1$, then $\varphi = (\alpha \circ \beta)$ for $\circ \in \{\wedge, \vee, \otimes, \Rightarrow, \sim, \Leftrightarrow\}$ and we have the following cases.
  - If $\varphi = (\alpha \wedge \beta)$, then we get $g(\min\{f_\alpha, f_\beta\}) = \max\{g(f_\alpha), g(f_\beta)\}$ and the claim follows by inductive hypothesis on α, β whose number of involved connectives is less or equal than k. The argument still holds replacing $\wedge$ with $\vee$ and min with max.
  - If $\varphi = (\alpha \otimes \beta)$, then we get

    $$g(g^{-1}(\min\{g(0^+), g(f_\alpha) + g(f_\beta)\})) = \\ = \min\{g(0^+), g(f_\alpha) + g(f_\beta)\}\ .$$

    As in the previous case, the claim follows by inductive hypothesis on α, β.
  - The remaining of the cases can be treated at the same way and noting that $\sim \alpha = \alpha \Rightarrow 0$.

□

The simplification property provides several advantages from an implementation point of view. On one hand it allows the evaluation of the generator function only on grounded predicate expressions and avoids an explicit computation of the pseudo-inverse g^{-1}. In addition, this property provides a general method to implement n-ary t-norms, of which universal quantifiers can be seen as a special case since we only deal with finite domains (see more in Section 4.3.2).

The simplification property yields an interesting analogy between truth functions and loss functions. In logic, the truth degree of a formula is obtained by combining the truth degree of its sub-formulas by means of connectives and quantifiers. At the same way, the loss corresponding to a formula that satisfies the property is obtained by combining the losses corresponding to its sub-formulas and connectives and quantifiers combine losses rather than truth degrees.

4.3 Logic and Learning

This Chapter presents a theoretical apparatus that may be exploited in different learning settings, especially in contexts where some relational knowledge about the task is available, and the input patterns are not assumed to be independent and identically distributed. According to the *learning from constraints* paradigm in Section 4.1 (from (Gnecco et al., 2015)), knowledge is represented by a set of constraints and the learning process is conceived as the problem of finding the task functions (implementing FOL predicates) that best satisfy the constraints. In particular in multi-task learning, additional information can be expressed by logical constraints, and supervisions are a special class of constraints forcing the fitting of the positive and negative examples for the task. An example for extra prior knowledge that may be available about a learning task, could be the statement 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.

Now, we brefly introduce how the logic knowledge describable using the proposed theory can be attached to a neuro-symbolic learning task. This will be much more deeply investigated in the following Chapter, where an entire programming framework implementing the theory will be shown. Let us consider a multi-task learning problem, where $\boldsymbol{p} = (p_1, \ldots, p_J)$ denotes the vector of real-valued functions (*task functions*) to be determined. Given the set $\mathcal{X} \subseteq \mathbb{R}^n$ of available data, a supervised learning problem can be generally formulated as $\min_{\boldsymbol{p}} \mathcal{L}(\mathcal{X}, \boldsymbol{p})$ where $\mathcal{L}$ is a positive-valued functional denoting a certain loss. In this framework, this setup is expanded assuming that the task functions are FOL predicates and all the available knowledge about these predicates, including supervisions, is collected into a knowledge base expressed via a set of FOL formulas $KB = \{\psi_1, \ldots, \psi_H\}$. The learning task is generally expressed as:

$$\min_{\boldsymbol{p}} \mathcal{L}(\mathcal{X}, KB, \boldsymbol{p}) \ .$$

The link between FOL knowledge and learning can be summarized as follows.

- Each *Individual* is an element of a specific domain, which can be used to ground the predicates defined on such domain. Any replacement of variables with individuals for a certain predicate is called *grounding*.

- *Predicates* express the truth degree of some property for an individual (unary predicate) or group of individuals (n-ary predicate).

- The *knowledge base* (KB) is a collection of FOL formulas expressing the learning task. The integration of learning and logical reasoning is achieved by compiling the logical rules into continuous real-valued constraints, which correlate all the defined elements and enforce some desired behaviour on them.

For a given rule in the KB, individuals, predicates, logical connectives and quantifiers can all be seen as nodes of an *expression tree*. The translation to 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 the corresponding feature representation of the individual to which the variable is currently assigned;

- visiting a *predicate* computes the output of the predicate with the current input groundings;

- visiting a *connective* combines the grounded predicate values by means of the real-valued operations associated to the connective;

- visiting a *quantifier* aggregates the outputs of the expressions obtained for the single individuals (variable groundings).

Thus, the compilation of the expression tree allows us to convert formulas into real-valued functions, represented by a computational graph, where predicate functions are composed by means of the truth-functions corresponding to connectives and quantifiers. Given a generic formula φ, we call the corresponding real-valued function its *functional representation* f_φ. This representation is tightly dependent on the particular choice of the translating t-norm. For instance, given two predicates p_1, p_2 and the formula $\varphi(x) = p_1(x) \Rightarrow p_2(x)$, the functional representation of φ in Łukasiewicz logic is given by $f_\varphi(x, \boldsymbol{p}) = \min\{1, 1 - p_1(x) + p_2(x)\}$.

A special note concerns *quantifiers* that have to be thought of as aggregating operators with respect to the predicate domains. For instance, according to Novak (Novák et al., 2012), that first proposed a fuzzy generalization of first–order logic, the *universal* and *existential quantifiers* may be converted as the infimum and supremum over a domain variable (or minimum and maximum when dealing with finite domains) that are common to any t-norm fuzzy logic. In particular, given a formula $\varphi(x)$

depending on a certain variable $x \in \mathcal{X}$, where $\mathcal{X}$ denotes the available samples for one of the involved predicates in φ, the semantics of the quantifiers are fuzzified as:

$$\psi = \forall x\, \varphi(x) \longrightarrow \qquad f_\psi(\mathcal{X}, \boldsymbol{p}) = \min_{x \in \mathcal{X}} f_\varphi(x, \boldsymbol{p}),$$
$$\psi = \exists x\, \varphi(x) \longrightarrow \qquad f_\psi(\mathcal{X}, \boldsymbol{p}) = \max_{x \in \mathcal{X}} f_\varphi(x, \boldsymbol{p}).$$

As shown in the next section, this quantifier translation is not well justified for all t-norms and the proposed method enables a more principled approach to perform this translation.

Once all the formulas in KB are converted into real-valued functions, their distance from satisfaction (i.e. distance from 1-evaluation) can be computed according to a certain decreasing mapping L expressing the penalty for the violation of any constraint. Assuming rule independence, learning can be formulated as the joint minimization over the single rules using the following loss function factorization:

$$\mathcal{L}(\mathcal{X}, KB, \boldsymbol{p}) = \sum_{\psi \in KB} \beta_\psi L\big(f_\psi(\mathcal{X}, \boldsymbol{p})\big) \tag{4.6}$$

where any β_ψ denotes the weight for the logical constraint ψ in the KB, which can be selected via cross-validation or jointly learned (Kolb et al., 2018; Marra et al., 2019a), f_ψ is the functional representation of the formula ψ according to a certain t-norm fuzzy logic and L is a decreasing function denoting the penalty associated to the distance from satisfaction of formulas, so that $L(1) = 0$. This proposed method will show that the selected semantics of the f_ψ converting a generic formula ψ and the choice of the L loss are intrinsically connected, and they can be both derived by the selection of a t-norm generator.

4.3.1 Example

Let's consider a simple multi-label classification task where the objects A, B must be detected in a set of input images $\mathcal{I}$, represented as a set of features. The learning task consists in determining the predicates $p_A(i), p_B(i)$, which return true if and only if the input image i is predicted to contain the object A, B, respectively. The positive supervised examples are provided as two sets (or equivalently their membership functions) $P_A \subset \mathcal{I}, P_B \subset \mathcal{I}$ with the images known to contain the object A, B, respectively. The negative supervised examples for A, B are instead provided as two sets $N_A \subset \mathcal{I}, N_B \subset \mathcal{I}$. Furthermore, the location where the images have been taken is assumed to be known, and a predicate $SameLoc(i_1, i_2)$ can be used to express the fact whether images i_1, i_2 have been taken in the same location. It is finally assumed that it is known as prior knowledge that two images taken in the same location are likely to contain the same object. The semantics of the above learning task can be expressed using FOL via the statement declarations shown in Table 4.3, where it was

$\forall i_1, i_2 : SameLoc(i_1, i_2) \wedge A(i_1) \Rightarrow A(i_2)$
$\forall i_1, i_2 : SameLoc(i_1, i_2) \wedge B(i_1) \Rightarrow B(i_2)$
$\forall i : P_A(i) \Rightarrow A(i) \wedge N_A(i) \Rightarrow \neg A(i)$
$\forall i : P_B(i) \Rightarrow B(i) \wedge N_B(i) \Rightarrow \neg B(i)$
$P_A(i10) = 1, \;\; P_A(i101) = 1, \;\; N_B(i11) = 1, \;\; P_B(i103) = 1$
$SameLoc(i23, i60) = 1$

Table 4.3: Example of the declarative a learning task expressed using FOL.

assumed that images $i23, i60$ have been taken in the same location and it holds that $P_A = \{i10, i101\}$, $P_B = \{i103\}$, $N_A = \{i11\}$ and $N_B = \emptyset$. The statements define the constraints that the learners must respect, expressed as FOL rules. Please note that also the fitting of the supervisions are expressed as constraints, .

Given a selection of t-norm generator g and a set of images I, this DFL program is compiled into the following optimization task:

$$\begin{aligned}\arg\min_{\boldsymbol{p}} \quad & \beta_1 \sum_{i \in P_A} g(p_A(i)) + \beta_2 \sum_{i \in N_A} g(1 - p_A(i)) + \\ & \beta_3 \sum_{i \in P_B} g(p_B(i)) + \\ & \beta_4 \sum_{(i_1, i_2) \in I_{sl}} \big[\max(0, g(p_A(i_1)) - g(p_A(i_2))\big] + \\ & \beta_5 \sum_{(i_1, i_2) \in I_{sl}} \big[\max(0, g(p_B(i_1)) - g(p_B(i_2))\big]\end{aligned}$$

where β_i is a meta-parameter deciding how strongly the i-th contribution should be weighted, I_{sl} is the set of image pairs having the same location $I_{sl} = \{(i_1, i_2) : SameLoc(i_1, i_2)\}$ and the first two elements of the cost function express the fitting of the supervised data, while the latter two express the knowledge about co-located images.

4.3.2 Discussion

The presented framework can be contextualized among a new class of learning frameworks, which exploit the continuous relaxation of FOL provided by fuzzy operators to integrate logic knowledge in the learning process (Diligenti et al., 2017; Serafini and Garcez, 2016a; Marra et al., 2019b). All these frameworks require the user to define all the operators of a given t-norm fuzzy logic. On the other hand, the presented framework requires only the generator to be defined. This provides several advantages like a minimum *implementation effort*, and an improved *numerical stability*. Indeed, it is possible to apply the generator only on grounded atoms by exploiting the simplification property and this allows to apply the non-linear operation (generator) to the atoms, whereas all compositions are performed via stable operators (e.g.

min,max,sum). On the contrary, the previous FOL relaxations correspond to an arbitrary mix of non-linear operators, which can potentially lead to numerically unstable implementations.

The presented framework provides a fundamental advantage in the integration with the tensor-based machine learning frameworks like TensorFlow (Abadi et al., 2016) or PyTorch (Ketkar, 2017). Modern deep learning architectures can be effectively trained by leveraging tensor operations performed via Graphics Processing Units (GPU). However, this ability is conditioned on the possibility of concisely express the operators in terms of simple parallelizable operations like sums or products over n arguments, which are often implemented as atomic operation in GPU computing frameworks and do not require to resort to slow iterative procedures. Fuzzy logic operator can not be easily generalized to their n-ary form. For example, the Łukasiewicz conjunction $T_L(x,y) = \max\{0, x+y-1\}$ can be generalized to n-ary form as $T_L(x_1, x_2, \ldots, x_n) = \max\{0, \sum_{i=1}^{n}(x_i) - n + 1\}$. On the other hand, the general SS t-norm for $-\infty < \lambda < 0$, $T_\lambda^{SS}(x,y) = (x^\lambda + y^\lambda - 1)^{\frac{1}{\lambda}}$, does not have any generalization and the implementation of the n-ary form must resort to an iterative application of the binary form, which is very inefficient in tensor-based computations. Previous frameworks like LTN and SBR had to limit the form of the formulas that can be expressed, or carefully select the t-norms in order to provide efficient n-ary implementations. However, the presented framework can express operators in n-ary form in terms of the generators. Thanks to the simplification property, n-ary operators for any Archimedean t-norm can always be expressed as $T(x_1, x_2, \ldots, x_n) = g^{-1}(\sum_{i=1}^{n} g(x_i))$.

Chapter 5

LYRICS

In this Chapter, we proposed LYRICS, a Tensorflow-based programming framework exploiting the t-norm theory of the previous chapter to integrate machine learning with logic programs describing constraints over the learning problem. In Section 5.1, we show the details of the framework, giving a general overview of its syntax and showing how to integrate elements of an external learning problem into the framework. Moreover, in Section 5.1.2, some design choices are discussed and some best practices introduced. Then, in Section 5.2, we describe multiple very different learning problems implemented in LYRICS, showing the extreme flexibility of the framework. In Section 5.3, we investigate how exploiting generated t-norm from a parametric family allow us to switch smoothly among very different t-norms, providing a very viable tool for compromising between convergence rate and stability, soft and hard penalization of constraint un-satisfaction. Finally, in Section 5.4, we show a very peculiar application of LYRICS in image to image translation tasks, showing how the use of logic descriptions allows a much cleaner and understandable description of generative adversarial learning tasks.

5.1 The framework

5.1.1 Integrating Logic and Learning

This Section presents how LYRICS defines an TensorFlow environment in which learning and reasoning can take place at the same time.

Domains and Individuals The definition of the knowledge in the LYRICS 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, or all the points of the plane in their (x, y) form. The domains are then filled with their "inhabitants", on which the learning and reasoning will be carried.

Domains of individuals allow users to provide data to the framework as tensors that represent the leaves of the computational graph. A Domain D_i is always bound to a tensor $X_i \in \mathbb{R}^{d_i \times r_i}$, where d_i denotes the number of individuals in the i-th domain and r_i denotes the dimension of the representation of the data in the i-th domain[1]. Thus, individuals correspond to rows of the X_i tensor. Individuals can be represented by both *constant* and *variable* feature tensors. By taking into account partially or totally variable features for the individuals, LYRICS allows to consider individuals as learnable objects too. For example, given two individuals *Marco* and *Michelangelo* bound to a constant and a variable tensor respectively, we may want to learn the representation of *Michelangelo* by exploiting some joint piece of knowledge (e.g. `fatherOf(Marco, Michelangelo) -> similarTo(Michelangelo, Marco)`).

In LYRICS, a domain can be defined using the `Domain` construct. It takes two arguments as input:

- `label`, it is a unique string identifier of a given label;
- `data`, it is a rank 2 tensor, where rows represent different individuals and columns range over their numerical features. If a domain is temporarily empty (e.g. it will be filled afterwards), then a 0 row tensor must be provided.

For example, a domain called *Points* can be defined as:

```
Domain(label='Points', data=data)
```

where *data* is the placeholder of the input data. The elements of a domain are a sort of "anonymous" individuals that are collectively processed. In Section 4.2, we underlined the double description of individuals, i.e. a symbolic label and a numerical feature representations. In LYRICS domains, the row index is the label, while the row values represent its feature representation.

Sometimes, we have domains of individuals with no feature representation but we still want to use them to express relational knowledge. For these cases, LYRICS provides a special construct, `RangeDomain`, which simply allows the user to define a certain number of individuals without the need to specify their features. It accepts two attributes:

- `label`, it is a unique string identifier of a given label;
- `max_range`, it is the number of individuals of this domain.

[1]Here, we assume that the feature representation is given by a vector. However, the system also allows the individuals to be represented by a generic tensor.

An *individual* of a domain can also be separately specified, and a specific behavior can be defined for it. In particular, specific individuals can be added to a domain using the `Individual` construct. It accepts as input attributes:

- `label`, it is a unique string identifier of this individual in the entire program;
- `domain`, it is the domain it belongs to;
- `value`, it is an (optional) feature representation.

For example, a point `'p0'` can be defined as follows:

```
Individual(label='p0', domain=('Points'), value=p0)
```

This allows the user to give more intelligible labels to some individual.

It is important to note that the feature representation of an individual can be omitted. This happens in two cases: *(i)* when we are dealing with some *label-only* reasoning task where we are not interested in the feature representation, *(ii)* when this representation is unknown. In this last case, a random *variable* representation is assigned to the individual and this variable will be optimized in the overall optimization problem in such a way as the constraint to be mostly satisfied.

Functions and Relations A LYRICS *function* can be defined to map elements from the input domains into an element of an output domain. In particular, a unary function takes as input an element from a domain and transforms it into an element of the same or of another domain, while an n-ary function takes as input n elements, mapping them into an output element of its output domain. For example, it is possible to define arithmetic functions to operate over number domains, encoding functions to transform elements of a domain into a latent space, etc. LYRICS allows also to define a set of *relations* or *predicates*, which are functions mapping elements of the input domains to truth values, as for example: *isCat(x)*, or $f(x) > 3$.

FOL functions allow the mapping between individuals of the input domains to an individual of the output domain, i.e. $f_i : D^f_{i_1} \times \cdots \times D^f_{i_m} \rightarrow D^f_i$, where $D^f_{i_1}, \ldots, D^f_{i_m}$ are the input domains and D^f_i is the output domain. On the other hand, FOL predicates allow to express the truth degree of some property for individuals of the input domains; i.e. $p_i : D^p_{i_1} \times \ldots \times D^p_{i_m} \rightarrow \{true, false\}$, where $D^p_{i_j}$ is the j-th domain of the i-th predicate. Functions and predicates are implemented using a TF architecture as explained in the following. If the graph does not contain any variable tensor (i.e. it is not parametric), then we say it to be *given*; otherwise it will contains variables which will be automatically learned to maximize the constraints satisfaction. In this last case, we say the function/predicate to be *learnable*. Learnable functions can be (deep) neural networks, kernel machines, radial basis functions, etc.

The evaluation of a function or a predicate on a particular tuple $x_1, \ldots, x_m$ of input individuals (i.e. $f_i(x_1, \ldots, x_m)$ or $p_i(x_1, \ldots, x_m)$) is said a *grounding* for the function or for the predicate, respectively. LYRICS, like related frameworks (**?**Serafini and Garcez, 2016b), follows a fully grounded approach, which means that all the learning and reasoning processes take place only once functions and predicates have been fully grounded over all the possible input tuples (i.e. on the entire Cartesian product of the correponding input domains).

Let us indicate as X_k the set of patterns in the domain D_k, then $\mathcal{X}_i^f = X_{i_1}^f \times \cdots \times X_{i_m}^f$ is the set of groundings of the i-th function. Similarly, $\mathcal{X}_i^p$ is the collection of groundings for the i-th predicate. Finally, $\mathcal{F}(\mathcal{X}) = \{f_1(\mathcal{X}_1^f), f_2(\mathcal{X}_2^f), \ldots, \}$ and $\mathcal{P}(\mathcal{X}) = \{p_1(\mathcal{X}_1^p), p_2(\mathcal{X}_2^p), \ldots\}$ are the outputs for all function and predicates over their corresponding groundings, respectively.

The `Function` construct accepts as input attributes:

- `label`, it is a unique string identifier of this function in the entire program;
- `domains`, they are the inputs domains of the function;
- `function`, it is an instance of a Function object, which is the Tensorflow implementation od the function. See Section 5.1.2 for a detailed description.

The following example defines a LYRICS "encoder" function:

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

where the FOL function is bound to its TF implementation, which in this case is the *CNNEncoder* function.

The `Relation` construct accepts as input attributes:

- `label`, it is a unique string identifier of this relation in the entire program;
- `domains`, they are the inputs domains of the relation;
- `function`, it is an instance of a Function object, which is the Tensorflow implementation od the function. See Section 5.1.2 for a detailed description.
- `features` it is a boolean which declares if this relation operates either on the features (features=*True*, default) or on the label only (features=*False*).

For example, a predicate A approximated by a neural network NN, taking as input the patterns in the domain *Points* can be defined as:

```
Predicate("A", domains=("Points"), function=NN)
```

Constraints Finally, it is possible to state the knowledge about the world by means of a set of *constraints*. Each constraint is a generic FOL formula using as atoms the previously defined functions and relations.

If formulas contains variables, they need to be existentially or universally quantified (closed formulas). Even though this is a best practice for describing constraints which describe the learning problem, sometimes one wants to simply check the validity of a formula. In this case open formulas (not quantified variables) are allowed and the system provides all the truth values for all the grounding belonging to the not quantified variables.

The `Constraint` construct accepts as inputs:

- `description`, it is a string containing the formula declaration;
- `weight`, it is a real value which weights the constraint in the overall optimization.

For instance, if we are given the domain *Animals* and two predicates *bird* and *flies* defined on it, the user can express the knowledge that all the birds fly by means of the constraint:

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

The `Constraint` construct is extremely general. By a correct definition of domains, functions and relations, any constraint can be implemented. However, it is often the case that functions and relations are mplemented by well-known learning models (e.g. neural networks, auto-encoders) for which TensorFlow provides already-built loss-functions or regularization functions. In these cases, LYRICS provides two ad-hoc constraints, namely `PointwiseConstraint` and `RegularizationConstraint`, which allows

Integration of learning and logical reasoning is achieved by translating logical expressions into continuous real-valued constraints. The logical expressions correlate the defined elements and enforce some desired behaviour on them.

Variables, functions, predicates, logical connectives and quantifiers can all be seen as nodes of an *expression tree*. The real-valued constraint is obtained by a post-fix visit of the expression tree, where the visit action builds the correspondent portion of computational graph. In particular:

- visiting a *variable* x_i substitutes the variable with the tensor X_i bound to the domain it belongs to;
- visiting a *function* or *predicate* corresponds to the grounding operation, where, first, the Cartesian product of the input domains is computed and, then, the TF models implementing those functions are evaluated on all gorundings (i.e. $f(\mathcal{X})$ or $p(\mathcal{X})$)

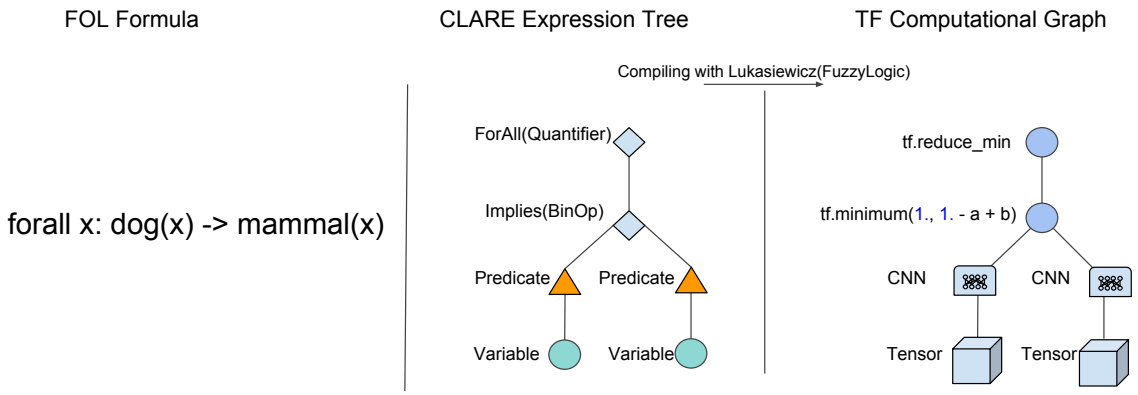


Figure 5.1: The translation of the FOL formula $\forall x\, dog(x) \rightarrow mammal(x)$ into a Lyrics expression tree and then its mapping to a TF computational graph.

- visiting a *connective* combines predicates by means of the real-valued operations associated to the connective by the considered t-norm fuzzy logic;
- visiting a *quantifier* aggregates the outputs of the expressions obtained for the single variable groundings.

Figure 5.1 shows the translation of a logic formula into its expression tree and successively into a TensorFlow computational graph.

A similar construct to the `Constraint` one is the `Query`. It receives as input a FOL formula, exactly like a constraint, but the compilation process is aimed at computing only the truth value of the corresponding query, without any interaction with the optimization process. Its use is commonly linked with the need to ask questions to the system after the learning process has been carried out.

Optimization Problem. As deeply described in Chapter 4, the theoretical framework implemented by LYRICS is aimed at designing an optimization process where a set of task functions (which are implementing `Function`s and `Predicate`s) is optimized so as to maximize the satisfaction of a collection of `Constraint`s.

The framework provides different ways to interact with an optimization routine:

- *Get the loss connected to each constraint.* Each `Constraint`, once compiled, returns the value of the loss correspondent to the chosen t-norm (or generator). The user is then allowed to exploit this loss in their own optimization process (for example, by integrating in an existing learning process).
- *Get the total constraint loss of Equation 4.6.* A unique pointer to a total loss is constantly incremented with the losses of every single added `Constraint` weighted by their parameter β_ψ.

- *Run a default training routine*. In this case, the actual constraint loss is added to all the pointwise constraints and all the regularization constraints into a unique objective. This objective is minimize by running a gradient descent scheme using Adam as parameter update algorithm. Learning rate, number of iterations and other optimization hyperparameters can be set by the user, in order to allow their validation.

5.1.2 Implementation Details

Functions API

Our framework reveals one of this primary strength in its declarativeness. The user is allowed to simply declare what the problem must do and it is the framework which takes care of implementing how it will do it. For this reason, all the algorithmic details, concerning how functions and predicates behaves and how they are optimized, are as hidden as possible.

To reach such degree of abstractness, the framework provides multiple pre-built constructs that the user is allowed to exploit in solving their problem. For example, the system provides already pre-built neural network classifiers for implementing trainable predicates, distance functions to implement given predicates, and a gradient descent optimization algorithm. All these constructs give a certain degree of customization; for example, the number and size of layers of a neural network.

Even though in many cases the provided constructs are sufficient, it is clear that a specific application could need some customization of these constructs.

In this document, we will show the *Function API*. This component of the framework provides both pre-built constructs and base classes for custom implementation and in this description we show both of them. This is important because, besides showing how to implement a custom function or predicate, we show some already implemented ones, allowing the user to understand the philosophy behind their design.

It is important to remember that the framework defines two different functional construct:

- `Relation`s, which are functions from domains to truth values, i.e. need to map inputs to a value between 0 and 1;

- `Function`s, which are functions from domains to domain, i.e. need to map inputs to the feature representation of an individual of a certain output domain.

When constructing both these objects, the user must provide the `function` attribute, which refers to the `Function` object which implements that relation or that function.

In this document we will show how the Function API is implemented and how the user can use it to define its own functions.

As a final remark, it is important to remember that this is the component of the framework which mostly depend on Tensorflow. Thus a deep understanding of this module requires some knowledge of Tensorflow, especially its low level API.

The Function class The base class of this module is the abstract class `Function`, which is implemented as follows.

```
class Function(object):
__metaclass__ = abc.ABCMeta

@abc.abstractmethod
def __call__(self, *a):
raise NotImplementedError()
```

This class exploits the `abc` module to be defined as *abstract* as in the OO meaning of the term, i.e. it defines at least one not implemented method (abstract method). Any object extending this class, must define a method `__call__` which represents the execution of this function on its inputs. This is a very natural definition of what a object representing a function would behave.

The reader is invited to look carefully at the fact that the method `__call__` of this class is not a generic one, but it is exactly the internal method `__call__` of the Python superclass `object`. This leads to a very nice feature of the Function API. Indeed, Python internally translate each function (as defined by the `def` operator) into a callable object (i.e. an object with an implemented `__call__` method). This means that every Python function, which uses only Tensorflow code, is a valid `Function` of our framework. Moreover, this also includes functions defined by means of `lambda` expressions.

For the sake of completeness, let us provide three examples on how to define functions in the framework. In particular, let us see how to implement a simple function adding two inputs.

- *Class Definition*

  ```
  class Add(Function):

  def __call__(self, a ,b):
  return tf.add(a,b)

  add = Add()
  ```

- *Function Definition*

```
def add_f(a ,b):
return  tf.add(a,b)

add = add_f
```

- *Lambda Expression*

```
add = lambda a,b: tf.add(a,b)
```

From a Python point of view, all three methods make `add` a callable object and, since only Tensorflow code is exploited, they are all valid ways of defining Functions in the framework.

Here, some examples on how to define FOL `Relation` and `Function` in the framework are provided, supposing the previous definition of a Domain `'Numbers'` representing real numbers.

```
Function(label='add',
         domains=('Numbers', 'Numbers'),
         function = Add())

Relation(label='equal',
         domains=('Numbers', 'Numbers'),
         function = lambda a,b: tf.cast(tf.equal(a,b), tf.float32))
```

About inputs to function When compiling formulas, the system will take care of calling the functions defined above on their inputs. However, it is extremely important, for users willing to design a new function, to know how inputs are shaped and passed to the function by the system.

It is really simple and it is based on some simple rules:

1. *The system works with linearized inputs* (as seen when Domains are described). This means that when providing data to the system the user is asked to provide them shaped as matrices with rows ranging over samples and columns ranging over features.

2. *The system provides linearized inputs to functions*. From the point above, the system will also call functions on linearized inputs. So each input to the function has to be thought as a a matrix with rows ranging over samples and columns ranging over features. The user pays attention that even with domains constituted by a unique feature the corresponding tensor will also have 2-sized shape $[M, 1]$ (i.e. it will be shaped as a matrix).

3. *The system takes care of performing the cartesian product for n-ary functions, with* $n > 1$. For n-ary functions, the system computes the cartesian product in advance and provides inputs already tupled. This extremely simplifies the task of the user which can then write the logic of the function on a ***per-row* basis**.

Let's provide an example. Consider the following function:

```
def isClose(a,b):
sqd = tf.squared_difference(a,b)
return tf.cast(tf.reduce_sum(sqd, axis=1)<0.5, tf.float32)
```

which is used to implement a relation which checks if two vectors `x` and `y` are close according to a squared euclidean distance. Then, consider the following program:

```
Domain(label='Points', data=X) # with X.shape = [M,N]
Relation(label='A', domains=('Points'), function=someOtherFunction)
Relation(label='isClose',
domains=('Points', 'Points'),
function=isClose)
Constraint('forall x: forall y: isClose(x,y) -> A(x) <-> A(y))')
```

The question we want to answer here is: when `'isClose(x,y)'` is compiled, which inputs `a` and `b` will the Function `isClose` be called on?

The answer is that both `a` and `b` will be two tensors of shape $[M \times M, N]$, obtained by repeating M times the rows of their domain tensor `X` according to the cartesian product order. This is induced by the fact that variables are always quantified, then all pairs need to be evaluated. This is illustrated in Figure 5.2

Thus, as the reader can observe, the function `isClose` can be defined on a per-row basis (i.e. `axis=1`) and this leads to very natural code for the functions. The user can avoid to take care of details which regard mostly the tensorial nature of the implementation.

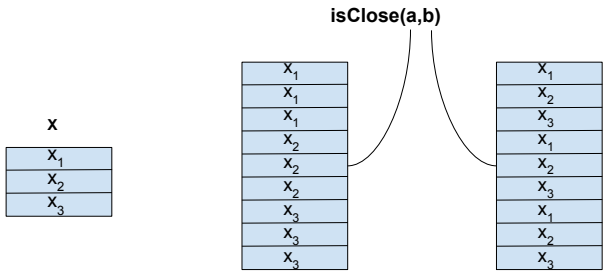


Figure 5.2: A picture illustrating how inputs get reshaped before being provided to the system

Learners and Regularized Learners The framework main construct `Constraint`, which guides both learning and inference, is paired with other two more specific constructs: `PointwiseConstraint` and `RegularizationConstraint`. These constructs are a shortcut to provide to the system faster ways to learn functions exploiting well-know deep learning loss functions without the need to explicity define them but simply using the tools provided by TensorFlow. We recall their syntax.

- *PointwiseConstraint*:

```
PointwiseConstraint(function=f, inputs=X, labels=y)
```

  where `f` is a special function, called *Learner*, which we will describe shortly, and `X` and `y` are tensors pairing inputs to their labels.

- *RegularizationConstraint*:

```
RegularizaionConstraint(function=f)
```

  where `f` is a special function, called *RegularizedFunction*.

As we pointed out, in order to enable the use of these special constructs, the `f` function to learn must belong to two special subclasses of `Function`, `Learner` and `RegularizedFunction`, for *PointwiseConstraint* and *RegularizationConstraint* respectively.

- **Learner**. A `Learner` is a abstract subclass of `Function`. As so, any instance of this class must declare a `__call__` method. Moreover, they must implement another method `cost` with the following signature:

```
def cost(self, labels, *inputs)
```

  A `PointwiseConstraint` acts by calling the `cost` method of its `function` attribute on its `inputs` and `labels` attributes.

- **RegularizedFunction**. A `RegularizedFunction` is a abstract subclass of `Function`. As so, any instance of this class must declare a `__call__` method. Moreover, they must implement another method `regularization_cost` with the following signature:

```
def regularization_cost(self)
```

  A `RegularizationConstraint` acts by calling the `regularization_cost` method of its `function` attribute. This method does not operate on any input, but only on the internal behaviour of the function (e.g. on its parameters by constraining their norm)

Indexing Functions In using the framework, it is common to define a `Domain` of individuals which do not have a feature representation. For example, this happens when we want to describe relational data, where individuals are simple identifiers and we know only some relationships among them.

Also in these cases the user would like to express the knowledge about these individuals and, therefore, it would like to define `Relation`s on them. How to handle this? The theoretical answer has to be found in statistical relational learning frameworks which exploit logic to define relationships (e.g. Markov Logic Network (MLN), Probabilistic Soft Logic (PSL), etc.). The implementation answer is, instead, the *Indexing Function* pattern.

In frameworks like MLN and PSL, it is clear that the atomic pieces of knowledge to reason about are *ground atoms*, i.e. relations with a specific tuple of constants as arguments (no variables). Then, each ground atom is attached to its truth values if it is known or to a trainable variable if it is unknown. *Indexing functions* simply collect all the truth values (known or variable) of a specific relation in a big tensor and, when called, they return only the values corresponding to the provided inputs. Their are a wrapper for a tensorial indexing function which allows the parallel retrivial of several variables given tensorial indices as inputs.

The signature of an indexing function can be described by this snippet:

```
class IndexingFunction(Function):

def __init__(self, tensor):
pass

def __call__(self, *idx):
pass
```

The only attribute of this class is likely to be the `tensor` which contains all the truth values for the relation and it is passed in the constructor. The `__call__` method uses tensorial indexing operations (like `tf.gather` or `tf.gather_nd`) to return only those truth values corresponding to input indices. The user must assure that inputs are integers (since the standard fixed data type of all the framework is float). Relations using indexing functions usually are defined on `RangeDomain`s or with `features=False` attribute (see the documentation about `Domain` and `Relation` to see how they construct integer tensors for indexing use.)

In the following, we will show some examples of IndexingFunction with their use in simple programs.

Binary Indexing Function with internal Variable. This binary indexing function is exploited when the relation is completely unknown and, thus, all its groundings

are mapped to trainable variables. Note the use of the sigmoid as thresholding function, to ensure the $[0,1]$ range. The variables are alla initialized to a value of -4, which corresponds nearly to a 0 value for the sigmoid (i.e. we are assuming a closed world for this relation).

```
class BinaryIndexFunction(Function):

def __init__(self, name, size_x=None,size_y=None):
super(BinaryIndexFunction, self).__init__()
self.var = tf.get_variable(name, initializer= -4 * tf.ones([size_x,size_y
    ]))

def __call__(self, idx1, idx0):
idx0 = tf.cast(idx0, tf.int64)
idx1 = tf.cast(idx1, tf.int64)
idx = tf.concat((idx1, idx0), axis=1)
res = tf.gather_nd(params=self.var, indices=idx)
return tf.sigmoid(res)
```

An example of a program using this function is shown. Here, we have the domain of *People*. There exists two relations.

```
Domain('People')
Individual('Marco', ('People'))
Individual('Giuseppe', ('People'))
Individual('Francesco', ('People'))
fo = BinaryIndexFunction(name='fo',size_x=n_people,sixe_y=n_people)
gfo = BinaryIndexFunction(name='gfo',size_x=n_people,sixe_y=n_people)

Relation('fatherOf', ('People', 'People'), fo, features=False)
Relation('grandFatherOf', ('People', 'People'), gfo, features=False)

Constraint('fatherOf(Marco,Giuseppe)')
Constraint('fatherOf(Giuseppe,Francesco)')
Constraint('forall x: forall y: forall z:
fatherOf(x,y) and fatherOf(y,z) -> grandFatherOf(x,z)')
```

Static Binary Indexing Function (with external Variable). This static binary indexing function is exploited when the tensor containing the truth values for every grounding is defined outside. This is the case when for example some values are known (and thus do not need to be optimized, are constant) while others are un-

known (need to be optimized, are variables). This allows the concatenation of the known values with the unknown variables outside the code.

```
class StaticBinaryIndexFunction(Function):

def __init__(self, var):
super(StaticBinaryIndexFunction, self).__init__()
self.var = var

def __call__(self, idx1, idx0):
idx0 = tf.cast(idx0, tf.int64)
idx1 = tf.cast(idx1, tf.int64)
idx = tf.concat((idx1, idx0), axis=1)
res = tf.gather_nd(params=self.var, indices=idx)
return res
```

Here the same example as before, even though this time fatherOf is completely known in advance and its values are passed from outside. In this case `'fatherOf'` will not be optimized but will be a given label-only predicate.

```
Domain('People')
Individual('Marco', ('People')) # id:0
Individual('Giuseppe', ('People'))  # id:1
Individual('Francesco', ('People')) # id:2

fatherOf_np = np.zeros([3,3])
fatherOf_np[0,1] = 1 # fatherOf(Marco,Giuseppe)
fatherOf_np[1,2] = 1 # fatherOf(Giuseppe,Francesco)
fo = StaticBinaryIndexFunction(var=fatherOf_np)
gfo = BinaryIndexFunction(name='gfo',size_x=n_people,sixe_y=n_people)

Relation('fatherOf', ('People', 'People'), fo, features=False)
Relation('grandFatherOf', ('People', 'People'), gfo, features=False)

Constraint('forall x: forall y: forall z:
fatherOf(x,y) and fatherOf(y,z) -> grandFatherOf(x,z)')
```

Equality Functions

- *Differentiable* Used to train functions or features on the basis of some similarities

```
    class L2SimilarityFunction(Function):
```

```
    def __call__(self, x, y):
    return 1 - tf.tanh(tf.reduce_sum(tf.squared_difference(x,y),axis
        =1))
```

- *Not Differentiable* Usually used to compare labels of individuals.

```
    class NotDifferentiableEqual():

    def __call__(self, a, b):
    return tf.cast(tf.equal(a,b), tf.float32)
```

TensorFlow Learners: FeedForward Neural Network Classifier Here we can see how to incorporate an entire TensorFlow model as a Learner.

```
class FFNClassifier(Learner):

def __init__(self, name, input_size, n_classes, hidden_layers = (10,)):
super(FFNClassifier, self).__init__()
self.name = name
self.output_size = n_classes
self.hidden_layers = hidden_layers
self.input_size = input_size
self._reuse = False

def _internal_(self,x):
with tf.variable_scope(self.name, reuse = self._reuse):
for hidden_size in self.hidden_layers:
x = tf.layers.dense(x, hidden_size, activation=tf.nn.sigmoid)
x = tf.layers.dense(x, self.output_size)
activation = tf.nn.softmax if self.output_size > 1 else tf.sigmoid
y = activation(x)
return y,x

def __call__(self,x):
x = tf.reshape(x, [-1, self.input_size])
y,logits = self._internal_(x)
self._reuse = True
return y

def cost(self, labels, input):
y, logits = self._internal_(input)
```

```
if self.output_size > 1:
loss = tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=
    labels)
else:
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=logits, labels=
    labels)
return tf.reduce_mean(loss)
```

5.2 Learning and Reasoning with LYRICS

This section presents a list of examples illustrating the range of learning tasks 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 and how to perform model checking. Moreover, we applied the proposed framework to two standard benchmarks: document classification in citation networks and term chunking in natural language text. The examples are presented using LYRICS syntax directly to show that the final implementation of a problem fairly retraces its abstract definition.

5.2.1 Semi-Supervised Learning

In this task we assume to have available a set of 420 points distributed along an outer and 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 5.3a. The remaining points are split into 200 unsupervised training points, shown in Figure 5.3b and 200 points left as test set. A neural network is assumed to have been created in TF to approximate the predicate A.

The network can be trained by making it fit the supervised data. So, 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("A", ("Points"), NN_A)
# Fit the supervisions
PointwiseConstraint(A, y_s, X_s)
```

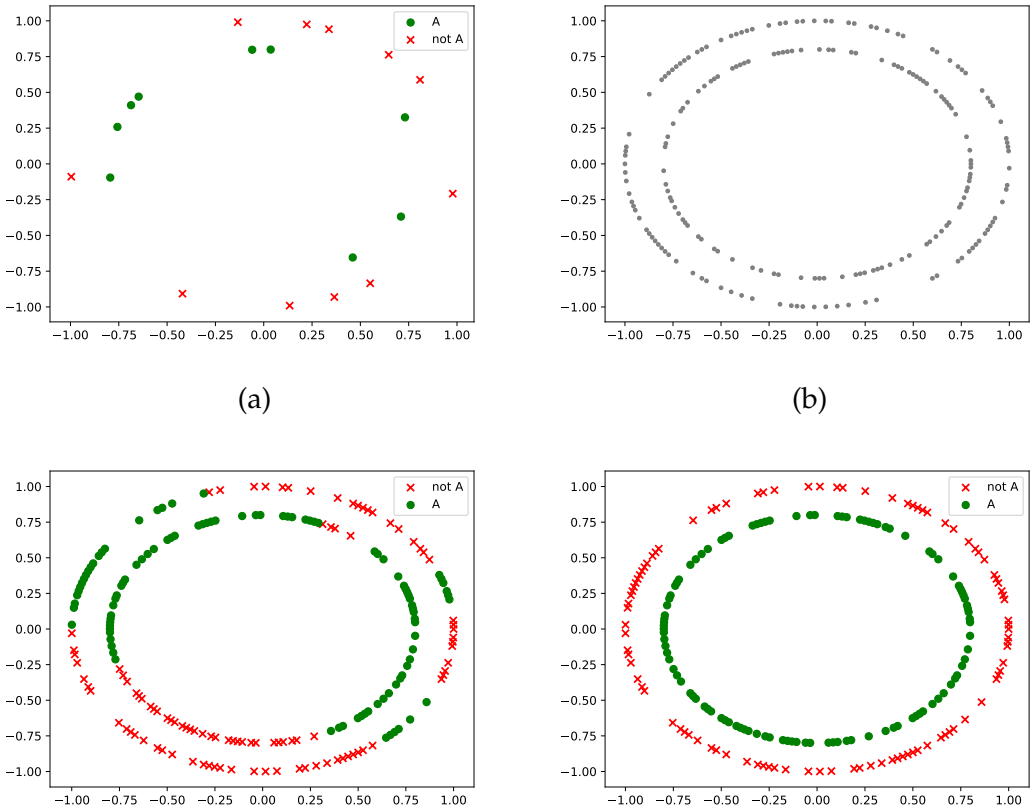


Figure 5.3: Semi-supervised Learning: (a) data that is provided with a positive and negative supervision for class A; (b) the unsupervised data provided to the learner; (c) class assignments using only the supervised examples; (d) class assignments using learning from examples and constraints.

Let's now assume that we want to express manifold regularization for the learned function: e.g. points that are close should be similarly classified. This can be expressed as:

```
# Predicate stating whether 2 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 5.3c shows the class assignments of the patterns in the test set, when using only classical learning from supervised examples. Finally, Figure 5.3d presents the

assignments when learning from examples and constraints.

5.2.2 Collective Classification

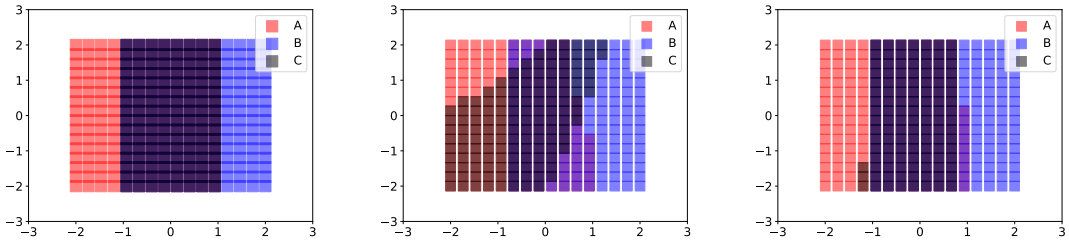
Collective classification (Sen et al., 2008) performs the class assignments exploiting any known correlation among the test patterns. This paragraph shows how to exploit these correlations in LYRICS. Here, we assume that the patterns are represented as $\mathbb{R}^2$ datapoints. The classification task is a multi-label problem where the patterns belongs 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 5.4a. 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)
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 5.4b. In a second stage, it is assumed that it is available some prior knowledge about the task at hand. 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 collective classification step is performed by seeking the class assignments that are close 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(A, priorsA,  X_test)
PointwiseConstraint(B, priorsB,  X_test)
PointwiseConstraint(C, priorsC,  X_test)
```



(a) (b) (c)

Figure 5.4: Collective classification: (a) classes assignments; (b) the predictions after the supervised step; (c) the predictions with collective classification and rules satisfaction (best viewed in colors).

where `X_test` the set of test datapoints and the outputs `priorsA`, `priorsB`, `priorsC` of the classifiers act as prior for the final assignments. As we can see from Fig.5.4c, the collective step fixes some wrong predictions.

5.2.3 Model checking

In this example, we show how the framework can be used to perform model checking. 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 using the vectors of supervisions y_A and y_B for the predicates A and B, respectively:

```
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 check if some given rule has been learned by the classifiers. To this hand, LYRICS allows to mark a set of constraints as test only, in order to perform model checking. In this case, constraints are only used to compute the degree of satisfaction of the corresponding FOL formulas over the data. For example, we checked the degree of satistaction of all possible formulas in Disjunctive Normal Form (DNF) that are universally quantified with a single variable. Only the constraint:

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

has a high truth degree (0.9997). As one could expect, the only fully-satisfied constraint (translated from DNF to its minimal form) is indeed $\forall x B(x) \rightarrow A(x)$, that states the inclusion of B in A. Model checking can be used as a fundamental step to perform rule deduction using the Inductive Logic Programming techniques (Muggleton and De Raedt, 1994).

5.2.4 Chunking

Given a sequence of words, term chunking (or shallow parsing) is a sequence tagging task aiming at linking constituent parts of sentences (nouns, verbs, adjectives, etc.) into phrases that form a single semantic unit. Following the seminal work by Collobert et al. (Collobert et al., 2011), many papers have applied deep neural networks to text chunking. Here, the deep learner is used to learn from examples as in classical supervised learning. Then we perform collective classification to fix some misclassification made by the network, according to certain logical rules expressing available prior knowledge.

We used the CoNLL 2000 shared task dataset (Tjong Kim Sang and Buchholz, 2000) to test the proposed methodology. The dataset contains 8936 training and 893 test English sentences. The task uses 12 different chunk types, which correspond to 22 chunk labels when considering the position modifiers. In particular, some labels have a B and I modifier to indicate for beginning and intermediate position in the chunk, respectively. For example, BVP indicates the start of a verbal phrase and IVP an intermediate term of the verbal phrase. The final performance is measured in terms of F1-score, computed by the public available script provided by the shared task organizers.

We selected the classifier proposed by Huang et al. (Huang et al., 2015) as our baseline, which is one of the best performers on this task. We used a variable portion of training phrases from the training set, ranging from 5% to 100%, to train the classifier, reusing the same parameters reported by the authors. The trained networks have then been applied on the test set providing an output score for each label for each term. It is well known that the output of the trained networks may not respect the semantic consistencies of the labels. For example, an intermediate token for a label must follow either a begin or intermediate one for the same label. For example, $\forall\, x \,\forall\, t\; BNP(x,t) \Rightarrow \neg IVP(x,t+1) \wedge \neg IPP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots$ expresses that if the t-th token is marked as the begin of a nominal phrase BNP the following token can not be an intermediate verbal IVP, intermediate prepositional IPP or intermediate adverbial $IADVP$ phrase. A small sample of the constraints stating

		% data in training set				
		5	10	30	50	100
F1	NN	87.39	89.55	92.15	93.31	94.18
	LYRICS	87.75	89.78	92.26	93.53	94.27
F1(rare tags)	NN	56.24	60.84	75.19	76.74	79.42
	LYRICS	57.65	61.36	75.68	77.45	79.71

Table 5.1: CoNLL2000 evaluation script on all the classes and on the less common pos tags that have an initial lower performance.

the output consistency can be expressed in FOL using the following statements:

$$
\begin{aligned}
&\forall\, x \,\forall\, t\; BNP(x,t) \Rightarrow \neg IVP(x,t+1) \wedge \neg IPP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; BVP(x,t) \Rightarrow \neg INP(x,t+1) \wedge \neg IPP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; BPP(x,t) \Rightarrow \neg IVP(x,t+1) \wedge \neg INP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; INP(x,t) \Rightarrow [\neg IVP(x,t+1) \wedge \neg IPP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; IVP(x,t) \Rightarrow [\neg INP(x,t+1) \wedge \neg IPP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; IPP(x,t) \Rightarrow [\neg IVP(x,t+1) \wedge \neg INP(x,t+1) \wedge \neg IADVP(x,t+1) \wedge \ldots \\
&\forall\, x \,\forall\, t\; INP(x,t+1) \Rightarrow BNP(x,t) \vee INP(x,t) \\
&\forall\, x \,\forall\, t\; IVP(x,t+1) \Rightarrow BVP(x,t) \vee IVP(x,t) \\
&\forall\, x \,\forall\, t\; IPP(x,t+1) \Rightarrow BPP(x,t) \vee IPP(x,t) \\
&\ldots
\end{aligned}
$$

where $P(x,t)$ indicates the output of the network associated to label P for the phrase x and the t-th term in the phrase.

In order to evaluate the proposed methodology, collective classification is performed to assign the labels in order to minimize the distance from the network outputs, acting as priors, while maximizing the verification of the constraints built from the previously reported rules. Table 5.1 reports the F1 results for the different percentages of supervised phrases used to train the network. The results have been evaluated both on all classes, and then zooming in for some of the rare classes that are often wrongly classified. The effect of the rules is overall mildly positive as most of the tags can be correctly predicted by the supervised examples. However, the effect of the knowledge is more clear when zooming in to see the effect on the some of the less common tags ($ADJ, ADV, PRT, SBAR$): since not enough examples are observed for these tags, the extra knowledge allows to improve their classification. Since these tags are relatively rare the overall effect on the metrics is not large on this dataset, but it is a very promising start to allow the application of pos tagging to challenging domains.

5.2.5 Document Classification on the Citeseer dataset

This section applies the proposed framework to a standard ML dataset. The CiteSeer dataset[2] (Lu and Getoor, 2003) consists of 3312 scientific papers, each one assigned to one of 6 classes: Agents, AI, DB, ML and HCI. The papers are not independent as they are connected by a citation network with 4732 links. Each paper in the dataset is described via its bag-of-word representation, which is a vector having the same size of the vocabulary with the i-th element having a value equal to 1 or 0, depending on whether the i-th word in the vocabulary is present or not present in the document, respectively. The dictionary consists of 3703 unique words. This learning task is expressed as:

```
Domain(label="Papers", data=X)
Predicate("Agents",("Papers"), Slice(NN, 0))
Predicate("AI",("Papers"), Slice(NN, 1))
Predicate("DB",("Papers"), Slice(NN, 2))
Predicate("IR",("Papers"), Slice(NN, 3))
Predicate("ML",("Papers"), Slice(NN, 4))
Predicate("HCI",("Papers"), Slice(NN, 5))
```

where the first line defines the domain of scientific articles to classify, and one predicate for each class is defined and bound to an output of a neural network NN, which features a softmax activation function on the output layer.

The domain knowledge that if a paper cites another one, they are likely to share the same topic, is expressed as:

```
Predicate("Cite",("Papers","Papers"),f_cite)
Constraint("forall x: forall y: Agent(x) and Cite(x, y) -> Agent(y)")
Constraint("forall x: forall y: AI(x) and Cite(x, y) -> AI(y)")
Constraint("forall x: forall y: DB(x) and Cite(x, y) -> DB(y)")
Constraint("forall x: forall y: IR(x) and Cite(x, y) -> IR(y)")
Constraint("forall x: forall y: ML(x) and Cite(x, y) -> ML(y)")
Constraint("forall x: forall y: HCI(x) and Cite(x, y) -> HCI(y)")
```

where `f_cite` is a given function determining whether a pattern cites another one. Finally, the supervision on a variable size training set can be provided by means of:

```
PointwiseConstraint(NN, y_s, X_s)
```

where `X_s` is a subset of the domain of papers where we enforce supervisions `y_s`.

Table 5.2 reports the accuracy obtained by a neural network with one hidden layer (200 hidden neurons) trained in a supervised fashion and by training the same network from supervision and logic knowledge in LYRICS, varying the amount of

[2] https://linqs.soe.ucsc.edu/data

	% data in training set				
	10	30	50	70	90
NN	60.08	68.61	69.81	71.93	72.59
LYRICS	**67.39**	**72.96**	**75.97**	**76.86**	**78.03**

Table 5.2: Citeseer dataset: comparison of the 10-fold average accuracy obtained by supervised training of a neural network (NN), and by learning the same NN from supervision and logic knowledge in LYRICS for a variable percentage of training data. Bold values indicate statistically significant improvements.

Method	Accuracy
Naive Bayes	74.87
ICA Naive Bayes	76.83
GS Naive Bayes	76.80
Logistic Regression	73.21
ICA Logistic Regression	77.32
GS Logistic Regression	76.99
Loopy Belief Propagation	77.59
Mean Field	77.32
NN	72.59
LYRICS	**78.03**

Table 5.3: Citeseer dataset: comparison of the 10-fold average accuracy obtained by content based and network based classifiers and by learning from supervision and logic knowledge in LYRICS.

available training data and averaged over 10 random splits of the training and test data. The improvements over the baseline are statistically significant for all the tested configurations. Table 5.3 compares the neural network classifiers against other two content-based classifiers, namely logistic regression (LR) and Naive Bayes (NB), and against collective classification approaches using network data: Iterative Classification Algorithm (ICA) (Neville and Jensen, 2000) and Gibbs Sampling (GS) (Lu and Getoor, 2003) both applied on top of the output of LR and NB content-based classifiers. Furthermore, the results against the two top performers on this task: Loopy Belief Propagation (LBP) (Sen et al., 2008) and Relaxation Labeling through Mean-Field Approach (MF) (Sen et al., 2008) are reported. The accuracy values are obtained as average over 10-folds created by random splits of size 90% and 10% of the overall data for the train and test sets, respectively. Unlike the other network based approaches that only be run at test-time (collective classification), LYRICS can distill the knowledge in the weights of the neural network. The accuracy results are the highest among all the tested methodologies in spite that the underlying neural

network classifier trained only via the supervisions did perform slightly worse than the other content-based competitors.

5.3 Investigating generated t-norms

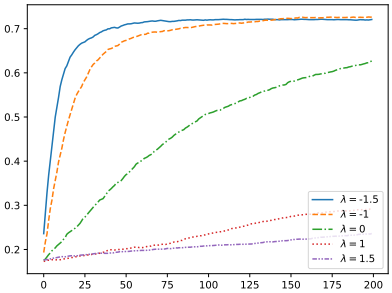
The experimental results have been carried out using the *Deep Fuzzy Logic* (DFL) software framework[3], which is an extension of LYRICS with generated t-norms. The formulas are compiled into a learning task using the theory of generators described in the previous Chapter. In the following of the section, it is assumed that each FOL constant corresponds to a tensor storing its feature representation.

5.3.1 The learning task

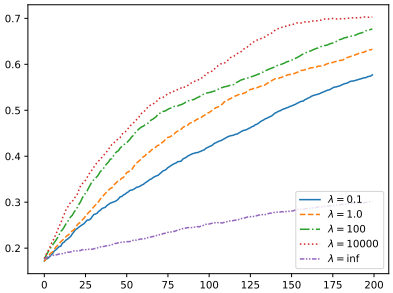
The CiteSeer dataset (Fakhraei et al., 2015) consists of 3312 scientific papers, each one assigned to one of 6 classes: Agents, AI, DB, IR, ML and HCI. The papers are not independent as they are connected by a citation network with 4732 links. This dataset defines a relational learning benchmark, where it is assumed that the representation of an input document is not sufficient for its classification without exploiting the citation network. The citation network is typically employed by assuming that two papers connected by a citation belong to the same category. This knowledge can be expressed by providing a general rule of the form: $\forall x \, \forall y \; cite(x,y) \Rightarrow \big(p(x) \iff p(y)\big)$ where *cite* is a binary predicate encoding the fact that x is citing y and p is a task function implementing the membership function of one of the six considered categories. This logical formula expresses a general concept called manifold regularization, which often emerges in relational learning tasks. Indeed, by linking the prediction of two distinct documents, the behaviour of the underlying task functions is regularized enforcing *smooth* transition over the manifold induced by the *cite* relation.

Each paper is represented via its bag-of-words, which is a vector having the same size of the vocabulary with the i-th element having a value equal to 1 or 0, depending on whether the i-th word in the vocabulary is present or not present in the document, respectively. The dictionary consists of 3703 unique words. The set of input document representations is indicated as X, which is split into a training and test set X_{tr} and X_{te}, respectively. The percentage of documents in the two splits is varied across the different experiments. The six task functions p_i with $i \in \{Agents, AI, DB, IR, ML, HCI\}$ are bound to the six outputs of a Multi-Layer-Perceptron (MLP) implemented in TF. The neural architecture has 3 hidden layers, with 100 ReLU units each, and softmax activation on the output. Therefore, the task functions share the weights of the hidden layers in such a way that all of them can exploit a common hidden representation. The *cite* predicate is a given (fully known

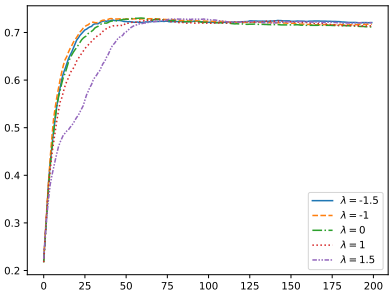
[3] http://sailab.diism.unisi.it/deep-logic-framework/



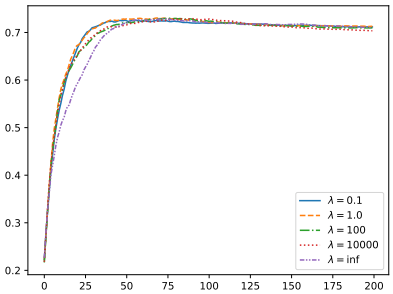
(a) SS - GD



(b) Frank - GD



(c) SS - Adam



(d) Frank - Adam

Figure 5.5: Learning Dynamics in terms of test accuracy on a supervised task when choosing different t-norms generated by the parameterized SS and Frank families. (5.5a) and (5.5b) are learning processes optimized with Vanilla Gradient Descent, while (5.5c) and (5.5d) are learning processes optimized with Adam Gradient Descent.

a prior) function, which outputs 1 if the document passed as first argument cites the document passed as second argument, otherwise it outputs 0. Furthermore, a given function P_i is defined for each p_i, such that it outputs 1 iff x is a positive example for the category i (i.e. it belongs to that category). A manifold regularization learning problem can be defined in DFL by providing, $\forall i \in \{Agents, AI, DB, IR, ML, HCI\}$, the following two rules:

$$\forall x \, \forall y \quad cite(x,y) \iff \big(p_i(x) \Rightarrow p_i(y)\big) \tag{5.1}$$

$$\forall x \quad P_i(x) \Rightarrow p_i(x) \tag{5.2}$$

where only positive supervisions have been provided because the trained networks for this task employ a softmax activation function on the output layer, which has the effect of imposing mutually exclusivity among the task functions, reinforcing the positive class and discouraging all the others. While this behaviour could have been trivially expressed using logic, this network architecture provides a principled baseline to compare against and it was therefore used across all the experiments for this dataset.

DLF allows the users to specify the weights of formulas, which are treated as hyper-parameters. Since we use at most 2 constraints per predicate, the β weight of the con-

straint expressing the fitting of the supervisions (Equation 5.2) is set to a fixed value equal to 1, while the weight of the manifold regularization rule expressed by Equation 5.1 is cross-validated from the grid of values $\{0.1, 0.01, 0.006, 0.003, 0.001, 0.0001\}$.

5.3.2 Results

Convergence rate

this experimental setup aims at verifying the relation between the choice of the generator and speed of convergence of the training process. In particular, a simple supervised learning setup is assumed for this experiment, where the leaning task is defined by Equation 5.2 by simply enforcing the fitting of the supervised examples. The training and test sets are composed of 90% and 10% of the total number of papers, respectively. Two parametric families of t-norms have been considered: the SS family (Definition 4.6) and the Frank family (Definition 4.7). Their parameter λ was varied to construct classical t-norms for some special values of the parameter but also to evaluate some intermediate ones. In order to keep a clear intuition behind the results, optimization was initially carried out using simple Gradient Descent with a fixed learning rate equal to $\eta = 10^{-5}$. Results are shown in Figures (5.5a) and (5.5b): it is evident that strict t-norms tend to learn faster than nilpotent ones by penalizing more strongly highly unsatisfied ground formulas. This difference is still remarkably present, although slightly reduced, by exploiting the state-of-the-art dynamic learning rate optimization algorithm Adam (Kingma and Ba, 2014) as shown in Figures 5.5c and 5.5d. This finding is consistent with the empirically well known fact that the cross-entropy loss performs well in supervised learning tasks for deep architectures, because it is effective in avoiding gradient vanishing in deep architectures. The cross-entropy loss corresponds to a strict generator with $\lambda = 0$ and $\lambda = 1$ in the SS and Frank families, respectively. This selection corresponds to a fast and stable converging solution when paired with Adam, while there are faster converging solutions when using a fixed learning rate.

Classification accuracy

the effect of the selection of the generator on classification accuracy is tested on a classification task with manifold regularization in the transductive setting, where all the data is available at training time, even if only the training set supervisions are used during learning. In particular, the data is split into different datasets, where $\{10\%, 25\%, 50\%, 75\%, 90\%\}$ of the available data is used a test set, while the remaining data forms the training data. During training, the fitting of the supervised data defined by Equation 5.2 can be applied only for the training data, while manifold regularization (Equation 5.1) can be enforced on all the available data. In this experiment, the Adam optimizer and the SS family of parametric t-norms have been

% Test	λ	Supervised		Manifold	
		Avg Accuracy	Stddev	Avg Accuracy	Stddev
10%	-1.5	72.44	0.8	79.07	1.07
	-1.0	72.26	0.96	79.37	0.68
	0.0	71.63	0.74	79.37	0.84
	1.0	71.57	0.88	78.58	0.69
	1.5	71.93	1.11	77.77	0.89
25%	-1.5	72.22	0.46	77.17	0.70
	-1.0	72.02	0.52	77.51	0.72
	0.0	71.35	0.56	77.39	0.50
	1.0	71.22	0.47	77.36	0.64
	1.5	71.51	0.77	76.41	0.57
50%	-1.5	70.94	0.56	75.52	0.46
	-1.0	70.98	0.51	76.16	0.32
	0.0	70.49	0.52	75.71	0.39
	1.0	70.07	1.71	76.39	0.46
	1.5	70.09	0.47	75.97	0.55
75%	-1.5	67.06	0.58	72.25	0.50
	-1.0	66.96	0.44	72.48	0.50
	0.0	67.02	0.54	72.73	0.61
	1.0	66.34	0.29	73.77	0.34
	1.5	65.93	0.64	73.37	0.37
90%	-1.5	61.09	0.78	66.02	2.51
	-1.0	61.59	0.44	67.24	1.72
	0.0	61.52	0.33	68.60	0.75
	1.0	61.31	0.52	70.69	0.52
	1.5	61.17	0.84	70.32	0.89

Table 5.4: Test accuracy of collective classification in transductive setting on the Citeseer dataset for different percentages of available training data and different selections of the parameter λ of the SS generator family.

employed. Table 5.4 shows the average test accuracy and its standard deviation over 10 different samples of the train/test splits. As expected, all generator selections improve the final accuracy over what obtained by pure supervised learning, as manifold regularization brings relevant information to the learner.

Table 5.4 also shows test accuracy when the parameter λ of the SS parametric family is selected from the grid $\{-1.5, -1, 0, 1, 1.5\}$, where values of $\lambda \leq 0$ move across strict t-norms (with $\lambda = 0$ being the product t-norm), while values greater than 0 move across nilpotent t-norms. (with $\lambda = 1$ being the Łukasiewicz t-norm). Strict t-norms seem to provide slightly better performances than nilpotent ones on

supervised task for almost all the test set splits. However, this does not hold in learning tasks using manifold regularization and a limited number of supervisions, where nilpotent t-norms perform better. An explanation of this behaviour can be found in the different nature of the two constraints, i.e. the supervision constraint of Equation 5.2 and the manifold regularization constraint of Equation 5.1. Indeed, while supervisions provide hard constraint that need to be strongly satisfied, manifold regularization is a general soft rule, which should allow exceptions. When the number of supervision is small and manifold regularization drives the learning process, the milder behaviour of nilpotent t-norms is better, as it more closely models the semantics of the prior knowledge. Finally, it is worth noticing that very strict t-norms (e.g. $\lambda = -1.5$ in the provided experiment) provide high standard deviations compared to other t-norms, especially in the manifold regularization setup. This shows the presence of a trade-off between the improved learning speed provided by strict t-norms and the instability due to their extremely non-linear behaviour.

Competitive evaluation

Table 5.5 compares the accuracy of the selected neural model (NN) trained only with supervised constraint against other two content-based classifiers, namely logistic regression (LR) and Naive Bayes (NB). These baseline classifiers have been compared against collective classification approaches using the citation network data: Iterative Classification Algorithm (ICA) (Neville and Jensen, 2000) and Gibbs Sampling (GS) (Lu and Getoor, 2003) applied on top of the output of the LR and NB content-based classifiers. Furthermore, the results are compared against the two top performers on this task: Loopy Belief Propagation (LBP) (Sen et al., 2008) and Relaxation Labeling through Mean-Field Approach (MF) (Sen et al., 2008). Finally, the results of DFL built by training the same neural network with both supervision and manifold regularization constraints, for which it was used a generator from the SS family with $\lambda = -1$. The accuracy values are obtained as an average over 10-folds created by random splits of 90% and 10% of the data for the train and test sets, respectively. Unlike the other relational approaches that can only be executed at inference time (collective classification), DFL can distill the knowledge in the weights of the neural network. The accuracy results are the highest among all the tested methodologies in spite of the fact that the neural network trained only on the supervisions performs slightly worse than the other content-based competitors.

5.4 Generative Learning with Logic

In the last few years the systematic adoption of deep learning to visual generation has produced impressive results that, amongst others, definitely benefit from the massive

Method	Classification Accuracy
Naive Bayes	74.87
ICA Naive Bayes	76.83
GS Naive Bayes	76.80
Logistic Regression	73.21
ICA Logistic Regression	77.32
GS Logistic Regression	76.99
Loopy Belief Propagation	77.59
Mean Field	77.32
NN	72.26
DFL	**79.37**

Table 5.5: Comparison of the accuracy on the Citeseer dataset obtained by content based and relational classifiers against supervised and relational learning expressed using DFL. All reported results are computed as average over 10 random splits of the train and test data. The bold number indicates the best performer and a statistically significant improvement over the competitors.

exploration of convolutional architectures. In this Section, we propose a general approach to visual generation where generation is driven by 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.

A special generation task is image-to-image translation, which learns to map each image for 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 (Rosca et al., 2017; Li et al., 2017) and outstanding results on image translation have been reported by (Zhu et al., 2017; Liu and Tuzel, 2016; Liu et al., 2017).

5.4.1 UNIT via logic description

This section shows 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. First, a set of *generator* functions $g_j : \mathcal{I} \rightarrow \mathcal{I}$, which take as input an image representation and generate a corresponding image in the same output domain, depending on the semantics given to the task. Second, a set of *discriminator* functions $d_i : \mathcal{I} \rightarrow [0, 1]$ determining whether an input image $x \in \mathcal{I}$ belongs to class i (i.e. stating if an image has got or not a given property) 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 as:

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

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 to transfer the knowledge provided by the supervision (i.e. the $S_i(x)$) inside the discriminators, which play a similar role. However, $d_i(x)$ functions are differentiable and can be exploited to train the generators functions. To this end, assuming that a given function has to generate 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 semantic of the j-th generator is to generate images of class j, this can be typically expressed by a rule taking the form:

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

In perspective, the logical formalism could provide a simple way to describe complex behavior of generator functions by interleaving multiple positive or negative discriminative atoms (i.e $d_i(g(x))$). By requiring that a given image should be classified as realistic, the GAN framework implements a special case of these constraints, where the required property is the similarity with real images.

Cycle consistency (Zhu et al., 2017) 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 to further restrict the number of possible translations. Assuming the semantic of the i-th generator is to generate images of class i, this can be naturally formulated as:

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

Clearly, in complex problems, the chain of functions intervening in these constraints can be longer.

The images in different domains are typically required to share the same latent space. Let us indicate $e : \mathcal{I} \rightarrow \mathbb{R}^n$ an encoding function mapping the image into a latent space. This encoding function must 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 using FOL as follows:

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

Up to now, the described constraints are very general and they can be exploited in almost all generative translation tasks. However, the logical formalism (and the LYRICS environment) allows the enforcement of any complex available knowledge about the task at hand. We will see some examples in the following experiment.

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, will produce an image of the next and previous digit, respectively. 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 output of such functions are still images, the output size of the networks is equal to the input size. A 1-hidden layer 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,

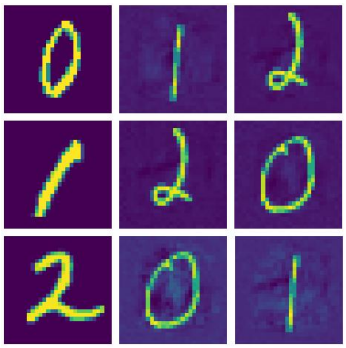


Figure 5.6: The first column pictures represents the input images. The second and third column pictures show the outputs of the functions `next` and `previous`, respectively, computed on the input image.

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 $S_i(x)$ in the general description.

The idea behind this task is to learn generative functions without giving any direct supervision to them, but simply requiring that the generation is consistent with the classification performed by some jointly learned classifiers. The problem can be described by the following constraints to learn the discriminators

$$\forall x\, isZero(x) \Rightarrow zero(x), \quad \forall x\, isOne(x) \Rightarrow one(x), \quad \forall x\, isTwo(x) \Rightarrow two(x)$$

and the following constraints to express that the generation functions are constrained to return images which are correctly recognized by the discriminators.

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

In addition, in order to force the generated images to be similar to at least one digit in the domain, we enforce the following constraints:

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

Finally, the cycle consistency constraints can be expressed by:

$$\forall x\, next(previous(x)) = x \qquad \forall x\, previous(next(x)) = x\ .$$

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 above constraints have been expressed in LYRICS and the model computational graphs have been bound to the predicates. Figure 5.6 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.

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, enable the construction of very complex generative problems by means of only an 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.

Figure 5.7: **Face Gender Translation: male to female.** The top row shows input male images, the bottom row shows the correspondent generated female images.

Figure 5.8: **Face Gender Translation: female to male.** The top row shows input female images, the bottom row shows the correspondent generated male images.

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

Experiments on Image Translation

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 (Zhu et al., 2017; Liu and Tuzel, 2016; Liu et al., 2017). In this section, we show how this 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 (Liu et al., 2015) 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.

In particular, we used the *Male* attribute to divide the entire dataset into the 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 the latent domain $\mathcal{Z} = \mathbb{R}^n$. The encoders are implemented as multilayer convolutional neural networks with resblocks (He et al., 2016), leaky-ReLU activation functions and instance normalization at each layer (see (Liu et al., 2017) 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 (Zhu et al., 2017; Liu and Tuzel, 2016; Liu et al., 2017). All these papers show that the use of convolutional models in conjunction with resblocks and instance normalization allows to obtain truly realistic and high definition images.

The problem can be described as follows. First, we look at the logical constraints the encoding and generation functions need to satisfy. We ask the encoder and 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. (Liu et al., 2017):

$$\forall x\ S_M(x) \Rightarrow g_M(e(x)) = x \tag{5.3}$$

$$\forall x\ S_F(x) \Rightarrow g_F(e(x)) = x \tag{5.4}$$

where the equality operator comparing two images in equations 5.3 and 5.4 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 pixel 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:

$$\forall x\ S_M(x) \Rightarrow g_M(e(g_F(e(x)))) = x \tag{5.5}$$

$$\forall x\ S_F(x) \Rightarrow g_F(e(g_M(e(x)))) = x \tag{5.6}$$

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:

$$\forall x\ S_M(x) \Rightarrow d_F(g_F(e(x))) \tag{5.7}$$
$$\forall x\ S_F(x) \Rightarrow d_M(g_M(e(x))) \tag{5.8}$$

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

$$\forall x\ S_M(x) \Rightarrow d_M(x) \wedge \neg d_F(g_F(e(x))) \tag{5.9}$$
$$\forall x\ S_F(x) \Rightarrow d_F(x) \wedge \neg d_M(g_M(e(x))) \tag{5.10}$$

Using logical constraints allows us to give a clean and easy formulation of the adversarial setting. These constraints force the generation function to generate samples that are categorized in the desired class by the discriminator. Moreover, the decoupling between the models, used to implement the functions and which can be inherited from the previous literature, and the description of the problem makes really straightforward to extend or transfer this setting.

We implemented this mixed logical and learning task using LYRICS. The Product t-norm was selected to define the underlying fuzzy logic problem. This selection of the t-norm is particularly suited for this task because, as shown earlier, it defines a cross-entropy loss on the output of the discriminators, which is the loss that was used to train these models in their original setup. The e, g_M, g_F functions are trained to the satisfaction of the constraints defined in Equations (5.3) to (5.8), while d_M and d_F are trained to satisfy Equations (5.9) and (5.10). Weight learning for the models was performed used the Adam optimizer with a fixed learning rate equal to 0.0001. Some male-to-female and female-to-male translations are shown in Figures 5.7 and 5.8 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,

$$\forall x\ S_M(x) \Rightarrow d_E(g_E(e(x)))$$
$$\forall x\ S_F(x) \Rightarrow d_E(g_E(e(x)))$$
$$\forall x\ S_E(x) \Rightarrow g_E(e(x)) = x$$
$$\forall x\ S_M(x) \wedge S_E(x) \Rightarrow d_E(g_F(e(x)))$$
$$\forall x\ S_F(x) \wedge S_E(x) \Rightarrow d_E(g_M(e(x)))$$
$$\forall x\ S_M(x) \wedge S_E(x) \Rightarrow g_E(e(g_F(e(x))) = g_F(e(x))$$
$$\forall x\ S_F(x) \wedge S_E(x) \Rightarrow g_E(e(g_M(e(x))) = g_M(e(x))$$
$$\forall x\ S_M(x) \wedge \neg S_E(x) \Rightarrow g_M(e(g_E(e(x))) = g_E(e(x))$$
$$\forall x\ S_F(x) \wedge \neg S_E(x) \Rightarrow g_F(e(g_E(e(x))) = g_E(e(x))$$

and to add the following for the discriminator:

$$\forall x\ S_E(x) \Rightarrow d_E(x)$$
$$\forall x\ S_M(x) \wedge \neg S_E(x) \Rightarrow \neg d_E(g_E(e(x)))$$
$$\forall x\ S_F(x) \wedge \neg S_E(x) \Rightarrow \neg d_E(g_E(e(x)))$$

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 5.9 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.

5.5 Conclusions

This Chapter introduced and analyzed LYRICS as a MiniMax Entropy model based on fuzzy potentials and a functional approximation of MAP inference. We can resume

Figure 5.9: **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.

the main contribution of the chapter as follows:

- the differentiability of fuzzy potentials allows the use of variational methods for inference;
- it proposes a more general exploitation of t-norms fuzzy theory for the description of the logic potentials (e.g. generated t-norms);
- it provides an actual programming framework implementing the theory;
- it shows the great expressivity of LYRICS in very different tasks;
- it shows improvements over standard neural networks when structure knowledge is known (e.g. chunking)
- it shows similar results with standard statistical relational learning approaches (LBP) stacked on top of classifiers (e.g. link prediction)

However, LYRICS still come with some limitations:

- the distribution is not trained (β must be validated);
- the propagation of information at the purely symbolic level needs to detach the networks (i.e. collective classification)

The DLM model proposed in the following Chapter provides a solution for those cases in which the previous limitations prevent users to apply LYRICS.

Chapter 6

Deep Logic Models

In Chapter 3, we introduced a large class of learning models based on the principle of MiniMax Entropy. In Section 3.5, we showed as MAP inference can be exploited not only as an inference scheme, but also as a atomic step of a larger learning algorithm. This choice is justified in the setting of Variational EM (Neal and Hinton (1998)).

In this Chapter, we propose DLM, a class of probabilistic graphical models aiming at bridging classical supervised learning techniques with logical reasoning. Indeed, DLMs are MiniMax Entropy models where potentials are defined exploiting fuzzy logic potentials (Chapter 4). MAP inference is the key ingredient to carry on the learning process with Fuzzy potentials being the key design choice to allow an easy and fast MAP inference step that can be carried on by simple gradient-based schemes.

6.1 Model

We indicate as $\boldsymbol{\theta}$ the model parameters, with $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\beta}\}$, and $\boldsymbol{x}$ the collection of input sensory data. We indicate by $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ the multivariate random variables corresponding to the ground atoms of a possible world , where $n > 0$ denotes the number of ground atoms. Here, in order to be able to exploit fuzzy logic potentials, we relax the boolean logic in the continuous case, thus $\boldsymbol{y} \in [0, 1]^n$.

A Deep Logic Model assumes that $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{w})$ is modeled via a MiniMax Entropy model, such that:

$$p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{w}) = \frac{1}{Z} \exp(\Phi_r(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) + \sum_c \beta_c \Phi_c(y)) \tag{6.1}$$

where Φ_r and Φ_c are two classes of potentials, whose description will be provided in the rest of the Section. Z is the classical *partition function*:

$$Z = \int_{\boldsymbol{y}'} \exp(\Phi_r(\boldsymbol{y}', \boldsymbol{x}, \boldsymbol{w}) + \sum_c \beta_c \Phi_c(y')) d\boldsymbol{y}'$$

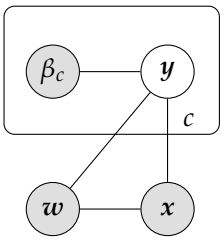


Figure 6.1: The DLM graphical model assumes the presence of two maximal cliques. The first one, concerning $\boldsymbol{\beta}$ and $\boldsymbol{y}$, models the behaviour of $\boldsymbol{y}$ independently from the sensorial data $\boldsymbol{x}$ by providing multiple constraints c. The second one, concerning $\boldsymbol{w}$, $\boldsymbol{x}$ and $\boldsymbol{y}$ models the behaviour of $\boldsymbol{y}$ when conditioned on a particular sensorial data $\boldsymbol{x}$.

Before going into more details, Figure 6.1 shows the graphical dependencies among the variables that are involved in our model. The first layer (Φ_r) processes the inputs using a model with parameters $\boldsymbol{w}$ and predicts $\boldsymbol{y}$. The second layer (Φ_c) takes as input $\boldsymbol{y}$ and applies reasoning using a set of constraints, whose parameters are indicated as $\boldsymbol{\beta}$.

6.1.1 Symbolic and Sub-symbolic reasoning

Deep Logic Models are conditional MiniMax Entropy models. By conditioning the distribution on a sensorial representation $\boldsymbol{x}$[1], we are enabling the possibility of exploiting MiniMax Entropy models for facing neuro-symbolic integration tasks.

In DLM, the *neural* and the *symbolic* components can be easily individuated in the two potentials exploited to design the distribution

The subsymbolic potential The sub-symbolic potential $\Phi_r(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w})$ is the responsible of the *neural* component, in the sense that it is expected to exploit neural computation to process raw/sensorial data to compute statistics over the variables of interests, i.e. $\boldsymbol{y}$.

In principle, one can think at a very general potential taking as inputs both $\boldsymbol{y}$ and $\boldsymbol{x}$ and to learn parameters $\boldsymbol{w}$ to provide a likelihood score to a given assignment of $\boldsymbol{y}$ given the observed data $\boldsymbol{x}$. This is the main role of the sub-symbolic potential. However, this unconstrained form, though correct, could be extremely complex to learn.

In DLM, we focus on a particular factorization of the Φ_r potential as:

$$\Phi_r(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) = -Dist(\boldsymbol{y}, f(\boldsymbol{x}; \boldsymbol{w})) \tag{6.2}$$

[1]Remember that $\boldsymbol{x}$ is a feature representation of the constants in a given world $\boldsymbol{y}$; i.e. $\boldsymbol{x} = g(\boldsymbol{y})$.

where $f(\boldsymbol{x}; \boldsymbol{w})$ is a generic neural network taking $\boldsymbol{x}$ as input and parametrized by the set of parameters $\boldsymbol{w}$. An example of $Dist(x, y)$ is the quadratic penalty $Dist(x, y) = \frac{1}{2}||x - y||^2$.

There are multiple reasons for the choice of this factorization that we show in turn. f is designed to be defined on the input space $\mathcal{X}$ and to return values in fuzzy n-dimensional simplex $[0, 1]^n$, i.e. $f : \mathcal{X} \to [0, 1]^n$. In other words, $f(\boldsymbol{x}; \boldsymbol{w})$ has the same dimensionality of $\boldsymbol{y}$. This creates a very strong link between the sub-symbolic part and the classical learning of neural networks. Indeed, f can be thought as a neural network trying to predict the ground atoms $\boldsymbol{y}$ by looking only at $\boldsymbol{x}$.

Φ_r is the only statistics of the DLM model that is learned by minimum entropy arguments, i.e. it is the only parametric potential. In other words, we are asking to the model to learn the neural network which predicts, given $\boldsymbol{x}$, the $\boldsymbol{y}$ which minimizes the entropy, i.e. the best neural network to maximize the likelihood.

This provides us with a very powerful inference tool. Indeed, we are enabling the system to have a double layer mechanism of inference. There is a *fast* inference mechanism (encoded inside the neural network), which allows a fast prediction of the ground atoms by simple association with the perceptual data. On top of this, there is the inference of the complete probabilistic model, which "corrects", i.e. improve, the initial predictions by asking coherence with the rest of the model (encoded inside the symbolic potentials Φ_c). This is an interpretation of DLM models that finds multiple links with current theories outside the computer science community about *intelligent thinking* (e.g. Kahneman (2011)).

The symbolic potentials Potentials Φ_c take care of the *symbolic* component of the reasoning. In fact, they only depend on the variables $\boldsymbol{y}$.

The DLM model focuses on potentials expressed on the space $\boldsymbol{y}$ by means of first–order logic (FOL) formulas, translated into real valued function by the t-norm theory. Instead of going into the details of this translation, which is identical to the one described in Chapter 4, we describe how this translation can be exploited to derive the symbolic potentials of the DLM model and which is the link with the general potentials of a MiniMax Entropy model. In particular, we want to see which is the general shape of a symbolic logic potential Φ_c which is responsible of providing the degree of satisfaction of $\boldsymbol{y}$ given a formula c. As seen in previous Chapters, the degree of satisfaction of a FOL formula c is obtained by:

- *Grounding the formula by substituting its variables with a given assignment of the constant.* In particular, let us define with k the number of variables of c. For any assignment A to the k variables of c with k constants, let $\gamma_{A,c}$ be the subset of ground atoms of $\boldsymbol{y}$ that are referred to the constants A and are present in c. For example, if we have three constants $C = \{\texttt{Giuseppe, Maria, Marco}\}$ and a

formula $c : \forall x \forall z \; \texttt{married}(x,z) \implies \texttt{sameFamily}(x,z)$, a possible assignment is $A = \{(x, \texttt{Giuseppe}), (z, \texttt{Maria})\}$. Than,

$$\gamma_{A,c} = \{y_{\texttt{married(Giuseppe,Maria)}}, y_{\texttt{sameFamily(Giuseppe,Maria)}}\}$$

- *Computing the truth degree of single groundings.* Let ϕ_c be the t-norm translation of the formula c up to the innermost quantifier. Than, for an assignment A and the restriction $\gamma_{A,c}$ of $\boldsymbol{y}$, we compute $\phi_c(\gamma_{A,c})$. In the previous case, given the Łukasiewicz t-norm, we have $\phi_c(\gamma) = \min(1, 1 - y_{\texttt{married\_x\_z}} + y_{\texttt{sameFamily\_x\_z}})$ and thus:

$$\phi_c(\gamma_{A,c}) = \min(1, 1 - y_{\texttt{married(Giuseppe,Maria)}} + y_{\texttt{sameFamily(Giuseppe,Maria)}})$$

- *Aggregating the truth degree of the formula over all groundings.* Let $\Gamma_c(\boldsymbol{y})$ the set of all the restrictions $\gamma_{A,c}$ of $\boldsymbol{y}$ for all possible assignments A to the k variables of c. The aggregation scheme is clearly defined by the quantifiers of the formula. Thus, the final shape of the subsymbolic potential is:

$$\Phi_c(\boldsymbol{y}) = \mathcal{A}\gamma_{A,c} \in \Gamma_c(\boldsymbol{y})[\phi_c(\gamma_{A,c})]$$

The choice of the notation is clearly not casual but retraces the one exploited in Chapter 3, Section 3.2.1, which the reader is invited to revise.

The link should be clear now. The number k of variables of a formula is indicating the size of the corresponding fragment. Quantifiers suggest the aggregation method, which clearly strongly influences the class of statistics our method evaluates. $\gamma_{A,c}$ is a special fragment which restricts $\boldsymbol{y}$ not only to a given choice of k constants but also to those relations indicated by the formula c.

Finally, we see that the general fragmentation principle, described in Section 3.2.1, whose *ratio* should be individuated in expressing global statistics as aggregation over local statistics, finds a natural case in quantified logical formulas.

6.1.2 Inference and Learning in DLM

MAP Inference The probability distribution modelled by a DLM is mostly designed for decision tasks, where the user is asked to take a decision on which assignment of $\boldsymbol{y}$ is mostly explaining the observed $\boldsymbol{x}$. In this tasks, the primary inference method is clearly MAP inference, which we now describe.

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

$$\boldsymbol{y}_M = \operatorname*{argmax}_{\boldsymbol{y}} \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{w}) = \operatorname*{argmax}_{\boldsymbol{y}} \left[\Phi_r(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) + \sum_c \beta_c \Phi_c(\boldsymbol{y}) \right] . \tag{6.3}$$

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

$$\nabla_y \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{w}) = \nabla_y \Phi_r(\boldsymbol{y}, \boldsymbol{x}, \boldsymbol{w}) + \sum_c \beta_c \nabla_y \Phi_c(\boldsymbol{y}) .$$

Learning As pointed out in Chapter 3, learning is framed into a MiniMax Entropy problem, which has a dual representation in a Maximum Likelihood problem. Thus, training can be carried out by maximizing the likelihood of the training data.

To this end, DLM takes a more Bayesian approch, thus maximizing the complete posterior over the parameters. Bayes theorem allows us to express this posterior as proportional to the posterior over $\boldsymbol{y}$ and the priors of the parameters; i.e.:

$$p(\boldsymbol{w}, \boldsymbol{\beta}|\boldsymbol{y}, \boldsymbol{x}) \propto p(\boldsymbol{y}|\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{x}) p(\boldsymbol{\beta}) p(\boldsymbol{w}) . \tag{6.4}$$

As pointed out in Section 3.5.4, the use of a Bayesian approach expressing priors over parameters leads to consequences which are extremely related to expressing a specific parsimony principle over the model (see Kaipio and Somersalo (2006) for more details in case of Gaussian priors).

Thus the maximization of the likelihood is:

$$\max_{\boldsymbol{\beta}, \boldsymbol{w}} \log p(\boldsymbol{w}, \boldsymbol{\beta}|\hat{\boldsymbol{y}}, \boldsymbol{x}) = \max_{\boldsymbol{\beta}, \boldsymbol{w}} \log p(\hat{\boldsymbol{y}}|\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{x}) + \log p(\boldsymbol{\beta}) + \log p(\boldsymbol{w}) \tag{6.5}$$

In particular, assuming that $p(\hat{\boldsymbol{y}}|\boldsymbol{w}, \boldsymbol{\beta}, \boldsymbol{x})$ follows the MiniMax Entropy model defined in Equation 6.1 and the parameter priors follow Gaussian distributions, yields:

$$\log p(\boldsymbol{w}, \boldsymbol{\beta}|\hat{\boldsymbol{y}}, \boldsymbol{x}) = -\frac{\alpha}{2}||\boldsymbol{w}||^2 - \frac{\lambda}{2}||\boldsymbol{\beta}||^2 - \Phi_r(\hat{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{w}) + \sum_c \beta_c \Phi_c(\hat{\boldsymbol{y}}) - \log Z ,$$

where α, λ are meta-parameters determined by the variance of the selected Gaussian distributions. Also in this case the likelihood may be maximized by gradient descent (Section 7.2.1) using the following derivatives with respect to the model parameters:

$$\begin{aligned} \frac{\partial \log p(\boldsymbol{\theta}|\hat{\boldsymbol{y}}, X)}{\partial \beta_c} &= -\lambda \beta_c + \Phi_c(\hat{\boldsymbol{y}}) - E_p\left[\Phi_c\right] \\ \frac{\partial \log p(\boldsymbol{\theta}|\hat{\boldsymbol{y}}, X)}{\partial w_i} &= -\alpha w_i + \frac{\partial \Phi_r(\hat{\boldsymbol{y}}, \boldsymbol{x}, \boldsymbol{w})}{\partial w_i} - E_p\left[\frac{\partial \Phi_r}{\partial w_i}\right] \end{aligned}$$

Unfortunately, as any other MiniMax Entropy model, the direct computation of the expected values in the above derivatives is not feasible. DLM can exploit the MAP inference approximation of Section 3.3.1, since as pointed out in the previous Section MAP inference is very fast due to the differentiability of the potentials. So, let $\boldsymbol{y}_M$ be the solution of the MAP inference problem (Equation 6.5).

The derivatives under the MAP approximation are:

$$\begin{aligned}\frac{\partial \log p(\boldsymbol{\theta}|\hat{\boldsymbol{y}},X)}{\partial \beta_c} &= -\lambda\beta_c + \Phi_c(\hat{\boldsymbol{y}}) - \Phi_c(\boldsymbol{y}_M)\\ \frac{\partial \log p(\boldsymbol{\theta}|\hat{\boldsymbol{y}},X)}{\partial w_i} &= -\alpha w_i + \frac{\partial \Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w})}{\partial w_i} - \frac{\partial \Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w})}{\partial w_i}\end{aligned}$$

We can think the MAP approximation to operate directly on the log-likelihood, thus maximizing an approximated objective function leading to the same derivatives. In particular, the approximated log-likelihood $\log \tilde{p}(\hat{\boldsymbol{y}}|\boldsymbol{x},\boldsymbol{\beta},\boldsymbol{w})$ is:

$$\log p(\hat{\boldsymbol{y}}|\boldsymbol{x},\boldsymbol{\beta},\boldsymbol{w}) \approx \log \tilde{p}(\hat{\boldsymbol{y}}|\boldsymbol{x},\boldsymbol{\beta},\boldsymbol{w}) \tag{6.6}$$

$$= \Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w}) - \Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w}) + \sum_c \beta_c \left(\Phi_c(\hat{\boldsymbol{y}}) - \Phi_c(\boldsymbol{y}_M)\right) \tag{6.7}$$

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 $\Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w}) \simeq \Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w})$ and $\Phi_c(\hat{\boldsymbol{y}}) \simeq \Phi_c(\boldsymbol{y}_M)\ \forall c$. Furthermore, the probability distribution is more centered around the MAP solution when $\Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w})$ is close to its maximum value. We assume that Φ_r is negative and have zero as upper bound: $\Phi_r(\boldsymbol{y},\boldsymbol{x},\boldsymbol{w}) \leq 0\ \forall \boldsymbol{y},\boldsymbol{x}$, like it holds for example for the already mentioned negative quadratic potential $\Phi_r(\boldsymbol{y},\boldsymbol{x},\boldsymbol{w}) = -\frac{1}{2}||\boldsymbol{y} - f(\boldsymbol{x},\boldsymbol{w})||^2$. Therefore, the constraint $\Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w}) \simeq \Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w})$ is transformed into the two separate constraints $\Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w}) \simeq 0$ and $\Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w}) \simeq 0$.

Therefore, given the current MAP solution, it is possible to increase the log likelihood by locally maximizing (one gradient computation and weight update) of the following cost functional: $\log p(\boldsymbol{w}) + \log p(\boldsymbol{\beta}) + \Phi_r(\hat{\boldsymbol{y}},\boldsymbol{x},\boldsymbol{w}) + \Phi_r(\boldsymbol{y}_M,\boldsymbol{x},\boldsymbol{w}) + \sum_c \beta_c \left[\Phi_c(\hat{\boldsymbol{y}}) - \Phi_c(\boldsymbol{y}_M)\right]$. In this Chapter, a quadratic form for the priors and the potentials is selected, but other choices are possible. 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:

$$\begin{aligned}C_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}},\boldsymbol{y}_M,X) = \frac{\alpha}{2}||\boldsymbol{w}||^2 + \frac{\beta}{2}||\boldsymbol{\beta}||^2 + \frac{1}{2}||\hat{\boldsymbol{y}} - f(\boldsymbol{x},\boldsymbol{w})||^2 +\\ +\frac{1}{2}||\boldsymbol{y}_M - f(\boldsymbol{x},\boldsymbol{w})||^2 + \sum_c \beta_c \left[\Phi_c(\hat{\boldsymbol{y}}) - \Phi_c(\boldsymbol{y}_M)\right]\ .\end{aligned}$$

Data: Input data X, output targets $\hat{\boldsymbol{y}}$, function models with weights $\boldsymbol{w}$
Result: Trained model parameters $\boldsymbol{\theta} = \{\boldsymbol{\beta}, \boldsymbol{w}\}$
Initialize $i = 0$, $\boldsymbol{\beta} = \mathbf{0}$, random $\boldsymbol{w}$;
while *not converged* $\wedge$ $i < max\_iterations$ **do**
  Compute function outputs $\boldsymbol{x}, \boldsymbol{w}$ on X using current function weights $\boldsymbol{w}$;
  Compute MAP solution $\boldsymbol{y}_M = \text{argmax}_{\boldsymbol{y}} \log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\beta}, \boldsymbol{w})$;
  Compute gradient $\nabla_{\boldsymbol{\theta}} C_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}}, \boldsymbol{y}_M, X)$;
  Update $\boldsymbol{\theta}$ via gradient descent: $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \beta_{lr} \cdot \nabla_{\boldsymbol{\theta}} C_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}}, \boldsymbol{y}_M, X)$;
  Set $i = i + 1$;
end

Algorithm 2: Iterative algorithm to train the function weights $\boldsymbol{w}$ and the constraint weights $\boldsymbol{\beta}$.

Minimizing $C_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}}, \boldsymbol{y}_M, X)$ is 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 $C_{\boldsymbol{\theta}}(\hat{\boldsymbol{y}}, \boldsymbol{y}_M, X)$ and one weight update step as summarized by Algorithm 2. For any constraint c, the parameter β_c admits also a negative value. This is in case the c-th constraint turns out to be also satisfied by the actual MAP solution with respect to the satisfaction degree on the training data.

6.2 Experimental Results

6.2.1 The PAIRS artificial dataset

Consider the following artificial task. We are provided with 1000 pairs of handwritten digits images sampled from the MNIST dataset. The pairs are not constructed randomly but they are compilied according to the following structure:

1. pairs with mixed even-odd digits are not allowed;
2. the first image of a pair represents a digit randomly selected from a uniform distribution;
3. if the first image is an even (resp. odd) digit, the second image of a pair represents one of the five even (resp. odd) digits with probabilities $p_1 \geq p_2 \geq p_3 \geq p_4 \geq p_5$, with p_1 the probability of being an image of the same digit, p_2 the probability of being an image of the next even/odd digit, and so on.

For example, if the first image of a pair is selected to be a *two*, the second image will be a *two* with probability p_1, it will be a *four* with probability p_2, a *six* with probability p_3 and so on, in a circular fashion. An example is shown in Figure 6.2. A correct classification happens when both digit in a pair are correctly predicted.

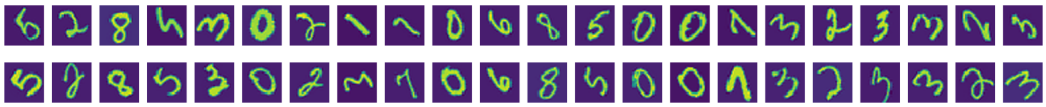


Figure 6.2: A sample of the data used in the PAIRS experiment, where each column is a pair of digits.

To model a task using DLMs there are some common design choices regarding these two features that one needs to take. We use the current example to show them. The first choice is to individuate the *constants* of the problem and their sensory representation in the perceptual space. Depending on the problem, the constants can live in a single or multiple separate domains. In the pairs example, the images are constants and each one is represented as a vector of pixel brightnesses like commonly done in deep learning.

The second choice is the selection of the *predicates* that should predict some characteristic over the constants and their implementation. In the pairs experiment, the predicates are the membership functions for single digits (e.g. `one(x)`, `two(x)`, etc.). A single neural network with 1 hidden layer, 10 hidden neurons and 10 outputs, each one mapped to a predicate, was used in this toy experiment. The choice of a small neural network is due to the fact that the goal is not to get the best possible results, but to show how the prior knowledge can help a classifier to improve its decision. In more complex experiments, different networks can be used for different sets of predicates, or each use a separate network for each predicate.

Finally, the *prior knowledge* is selected. In the pairs dataset, where the constants are grouped in pairs, it is natural to express the correlations among two images in a pair via the prior knowledge. Therefore, the knowledge consists of 100 rules in the form $\forall(x,y)\ D_1(x) \rightarrow D_2(y)$, where (x,y) is a generic pair of images and (D_1, D_2) range over all the possible pairs of digit classes.

We performed the experiments with $p_1 = 0.9, p_2 = 0.07, p_3 = p_4 = p_5 = 0.01$. All the images are rotated with a random degree between 0 and 90 anti-clockwise to increase the complexity of the task. There is a strong regularity in having two images representing the same digit in a pair, even some rare deviations from this rule are possible. Moreover, there are some inadmissible pairs, i.e. those containing mixed even-odd digits. The train and test sets are built by sampling 90% and 10% image pairs.

The results provided using a DLM have been compared against the following baselines:

- the same neural network (NN) used by DLM but with no knowledge of the structure of the problem;
- Semantic Based Regularization/Logic Tensor Networks (SBR/LTN), which

Model	NN	SBR	DLM-NN	DLM
Accuracy	0.62	0.64	0.65	0.76

Table 6.1: Comparison of the accuracy metric on the PAIRS dataset using different models.

are equivalent on this specific task. These frameworks employ the logical rules to improve the learner but the rule weights are fixed parameters, which are not jointly trained during learning. Since searching in the space of these parameters via cross-validation is not feasible, a strong prior was provided to make SBR/LTN prefers pairs with the same image using 10 rules of the form $\forall(x,y)\ D(x) \rightarrow D(y)$, for each digit class D. These rules hold true in most cases and improve the baseline performance of the network.

Table 6.1 shows how the neural network output of a DLM (DLM-NN) already beats both the same neural model trained without prior knowledge and SBR. This happens because the neural network in DLM is indirectly adjusted to respect the prior knowledge in the overall optimization problem. When reading the DLM output from the MAP solution (DLM), the results are significantly improved.

6.2.2 Link Prediction in Knowledge Graphs

Neural-symbolic approaches have been proved to be very powerful to perform approximated logical reasoning Trouillon et al. (2016). A common approach is to assign to each logical constant and relation a learned vectorial representation Bordes et al. (2013). Approximate reasoning is then carried out in this embedded space. Link Prediction in Knowledge Graphs is a generic reasoning task where it is requested to establish the links of the graph between semantic entities acting as constants. Rocktaschel et al. Rocktäschel and Riedel (2017) shows state-of-the-art performances on some link prediction benchmarks by combining Prolog backward chain with a soft unification scheme.

This section shows how to model a link prediction task on the *Countries* dataset using a Deep Logic Models, and compare this proposed solution to the other state-of-the-art approaches.

Dataset. The Countries dataset Bouchard et al. (2015) consists of 244 countries (e.g. germany), 5 regions (e.g. europe), 23 sub-regions (e.g. western europe, northern america, etc.), which act as the constants of the KB. Two types of binary relations among the constant are present in the dataset: $\texttt{locatedIn}(c_1, c_2)$, expressing that c_1 is part of c_2 and $\texttt{neighborOf}(c_1, c_2)$, expressing that c_1 neighbors with c_2. The knowledge base consists of 1158 facts about the countries, regions and sub-regions, expressed

in the form of Prolog facts (e.g. `locatedIn`(italy,europe)). The training, validation and test sets are composed by 204, 20 and 20 countries, respectively, such that each country in the validation and test sets has at least one neighbor in the training set. Three different tasks have been proposed for this dataset with an increasing level of difficulty. For all tasks, the goal is to predict the relation $\texttt{locatedIn}(c, r)$ for every test country c and all five regions r, but the access to training atoms in the KB varies, as explained in the following:

- Task S1: all ground atoms $\texttt{locatedIn}(c, r)$, where c is a test country and r is a region, are removed from the KB. Since information about the sub-region of test countries is still contained in the KB, this task can be solved exactly by learning the transitivity of the `locatedIn` relation.
- Task S2: like S1 but all grounded atoms $\texttt{locatedIn}(c, s)$, where c is a test country and s is a sub-region, are removed. The location of test countries needs to be inferred from the location of its neighbors. This task is more difficult than S1, as neighboring countries might not be in the same region.
- Task S3: like S2, but all ground atoms $\texttt{locatedIn}(c, r)$, where r is a region and c is a training country with either a test or validation country as a neighbor, are removed. This task requires multiple reasoning steps to determine an unknown link, and it strongly exploits the sub-symbolic reasoning capability of the model to be effectively solved.

Model. Each country, region and sub-region corresponds to a constant. Since the constants are just symbols, each one is assigned to an embedding, which is learned together with the other parameters of the model. The predicates are the binary relations `locatedIn` and `neighborOf`, which connect constants in the KB. Each relation is learned via a separate neural network with a 50 neuron hidden layer taking as input the concatenation of the embeddings of the constants. In particular, similarly to Bordes et al. (2013), the constants are encoded into a one-hot vector, which is processed by the first layer of the network, outputting an embedding composed by 50 real number values. As commonly done in link prediction tasks, the learning process is performed in a transductive mode. In particular, the input X consists of all possible constants for the task, while the train examples $\hat{\boldsymbol{y}}$ will cover only a subset of all the possible grounded predicates, leaving to the joint train and inference process the generalization of the prediction to the other unknown grounded relations. Indeed, the output of the train process in this case is both the set of model parameters and the MAP solution predicting the unknown grounded relations that hold true.

Multi-step dependencies among the constants are very important to predict the existence of a link in this task. For example in task S1, the prediction of a link among a country and a region can be established via the path passing by a sub-region,

Task	ComplEx	NTP	NTPβ	DLM
S1	99.37	90.83	**100.00**	**100.00**
S2	87.95	87.40	93.04	**97.79**
S3	48.44	56.68	77.26	**91.93**

Table 6.2: Comparison of the accuracy provided by different methods on link prediction on the Countries dataset. Bold numbers are the best performers for each task.

once the model learns a rule stating the transitivity of the `locatedIn` relation (i.e. $\texttt{locatedIn}(x,y) \wedge \texttt{locatedIn}(y,z) \rightarrow \texttt{locatedIn}(x,z)$). Exploiting instead the rule $\texttt{neighborOf}(x,y) \wedge \texttt{locatedIn}(y,z) \rightarrow \texttt{locatedIn}(x,z)$, the model should be capable of approximately solving task S2.

All 8 rules $\forall x\ \forall y\ \forall z\ \texttt{A}(x,y) \wedge \texttt{B(y,z)} \rightarrow \texttt{C}(y,z)$, where `A`, `B` and `C` are either `neighborOf` or `locatedIn` are added to the knowledge base for this experiment. These rules represent all the 2-steps paths reasoning that can be encoded, and the strength of each rule needs to be estimated as part of the learning process for each task. The training process will iteratively minimize Equation 6.8 by jointly determining the embeddings and the network weights such that network outputs and the MAP solution will correctly predict the training data, while respecting the constraints on the MAP solution at the same level as on the train data.

Results. Table 6.2 compares DLM against the state-of-the-art methods used by Rocktaschel et al. Rocktäschel and Riedel (2017), namely ComplEx, NTP and NTPβ. Task S1 is the only one that can be solved exactly when the transitive property of the `locatedIn` relation has been learned to always hold true. Indeed, most methods are able to perfectly solve this task, except for the plain NTP model. DLM is capable perfectly solving this task by joining the logical reasoning capabilities with the discriminative power of neural networks. DLMs perform better than the competitors on tasks S2 and S3, thanks to additional flexibility obtained by jointly training the relation functions using neural networks, unlike the simple vectorial operations like the cosine similarity employed by the competitors.

6.3 Conclusions

This Chapter presented DLMs, MiniMax Entropy models, where the use of fuzzy logic potentials allows for variational MAP inference.

The main contributions of this Chapter can be resumed as follows:

- DLMs can learn the relative weight (β) of each logical potential, which cannot be done in very similar approaches (i.e. SBR, LTN, LYRICS).

- As opposed to LYRICS, the neural component represent an initial (fast) starting point to fully symbolic inference to take place
- DLMs can improve w.r.t. to both neural networks (by allowing the use of structure) and to related approaches by being capable of weighting correctly each of the (possibly large number of) pieces of logical knowledge
- DLMs compare favourably also w.r.t. state-of-the-art neuro-symbolic approaches (Neural Theorem Provers and ComplEx) in link prediction tasks.

However, some improvements are still in order:

- Logic Knowledge, in the shape of FOL formulas, must be provided in advance. The disadvantages of this issue have been discussed in Chapter 3.
- MAP inference can be a rough approximation in not-decision-oriented tasks (e.g. generative tasks)

NMLNs, introduced in the next Chapter, will provide a solution to both issues.

Chapter 7

Neural Markov Logic Networks

In this Chapter, we exploit the MiniMax Entropy framework to propose Neural Markov Logic Networks (NMLN). Here, the statistics which are used to model the probability distribution are not known in advance, but are modelled as neural networks trained together with the probability distribution model. This is extremely powerful when compared to classical MLNs, where either domain experts are required to design some useful statistics about the domain of interest by hand (i.e. logical rules) or structure learning based on combinatorial search needs to be performed. These requirements normally limit a wide application of these models as out-of-the box tools. It is worth noticing that overtaking the need of such "feature-engineering" is one of the reasons behind the massive adoption of deep learning techniques. However, not much has been done in the same direction by the statistical relational learning community. Moreover, designing statistics as neural networks allows a more fine-grained description of the data, opening the doors to applications of our model to the generative setting.

In this Chapter, *(i)* we introduce a new statistical relational model, which overcomes actual limitations of both classical and recent related models such as (Richardson and Domingos, 2006; Rocktäschel and Riedel, 2017; Sourek et al., 2018); *(ii)* we propose a theoretical justification of the model as naturally emerging from a principle of Min-Max-entropy; and *(iii)* we showcase its effectiveness on two quite diverse problems: knowledge-base completion and generative modelling of small molecules.

7.1 Neural Markov Logic Networks

7.1.1 The Model

Neural Markov Logic Networks are a direct application of the MiniMax Entropy principle introduce in Chapter 3.

In particular, here we are not interested in modeling the link between the sub-

symbolic reasoning operating on the perceptual level and the symbolic reasoning operating on the abstract level as in the other Chapters. Instead, the integration of subsymbolic techniques and symbolic techniques is reached by exploiting models usually exploited for subsymolic reasoning (i.e. neural networks) to learn potentials on the symbolic level. In other words, the logic layer of the previous Chapter is completely substituted with a more complex (but expressive) neural layer.

The probability distribution $p(\omega)$ takes the following shape:

$$p(\omega; w) = \frac{1}{Z} \exp(\beta \Phi(\omega; w)) = \frac{1}{Z} \exp(\beta \sum_{\gamma \in \Gamma_k(\omega)} \phi(\gamma; w)) \tag{7.1}$$

Here, we assume no perceptual data x is conditioning the distribution. We assume the factorization of the potential $\Phi(\omega; w)$ as a sum over anonymized fragments $\phi(\gamma; w)$ as in Section 3.2.1.

Now, that we have discussed logical aggregators and quantifiers, it is easy to see the analogy between all the groundings of a formula with k quantified variables and all the fragments of size k of a possible world. When restricting the size of a fragment, we are focusing on local properties (i.e. local rules) to model the probability distribution. The connection between potentials in NMLN and logical rules is also analyzed in Section 7.1.4, where we show the clear link between a NLML and a standard Markov Logic Network.

7.1.2 Vector Embeddings of Domain Elements

By anonymizing a fragment, the model loses any trace of the identity of the constants involved in it, preserving only their structural behaviours. While this feature is essential to allow the identification of structural patterns also inside a single possible world, it prevents the model from having different behaviour on specific constants. This, instead, is a basic feature of many existing transductive models, like NTP (Rocktäschel and Riedel, 2017), which exploit the geometry of a latent representation space of constants to improve their prediction capabilities.

To this end, we define an *embedding fragmented neural potential* $\phi_e(\gamma, \widehat{S}; \mathbf{w}, \Theta)$, which is function of the anonymized fragment but also of the specific constants involved in it (i.e. the list of constants $\widehat{S}$). In particular, in transductive settings, we always have a possible world $\widehat{\omega}$ and we use the same constant set S both during learning and inference. Let $\Theta \in \mathbb{R}^{n \times d}$ be a variable embedding matrix. It can be considered a map from the constant set S to a latent real domain $\mathbb{R}^d$, i.e. the embedding space. Let $c(\widehat{S}, \Theta)$ be a function that concatenates the k rows of Θ corresponding to the k constants in the restricted set $\widehat{S}$. Thus, the embedding fragmented neural potential ϕ_e can be seen as a function of both γ, which encodes the structural properties of the fragment and $c(\widehat{S}, \Theta)$, which encodes the identity of constants by providing a

latent representation for them. In other words, $\phi_e(\gamma, \widehat{S}; \mathbf{w}, \Theta) = f(\gamma, c(\widehat{S}, \Theta); \mathbf{w})$ for some neural function f parameterized by $\mathbf{w}$. This is inspired by works in the NLP community (Mikolov et al., 2013; Mnih and Kavukcuoglu, 2013), where the c function can have different forms than concatenation. The components of the embedding vectors are treated as any other weights of the potential functions and are updated using gradients computed according to Section . Intuitively, the contrastive nature of the learning (Bordes et al., 2013; Trouillon et al., 2017), leads to the development of similar embeddings for similar constants. As we show in Section 7.2.2, the addition of embedding of constants helps improving the prediction capability of our model in transductive settings.

7.1.3 Inference

In order to design an optimization procedure to learn Neural Markov Logic Networks, we need to rely on some methods to sample from the distribution. In this Chapter, we exploit MCMC methods, in particular approximate Gibbs Sampling (Robert and Casella, 2013), to sample from Neural Markov Logic Networks. The approximation comes from the fact that GS requires a large number of steps before converging to the target distribution. However, we run it only for a limited number of steps t, which, in some cases, is restricted to $t = 1$. When this happens, our method recovers a discrete version of the Contrastive Divergence (CD) algorithm (Hinton, 2002).

Gibbs sampling cannot effectively handle distributions with a lot of determinism. In normal Markov logic networks, sampling from such distributions may be tackled by an algorithm called MC-SAT (Poon and Domingos, 2006). However, MC-SAT requires an explicit logical encoding of the deterministic constraints, which is not available in Neural Markov Logic Networks where deterministic constraints are implicitly encoded by the potential functions. In fact, only constraints that are almost deterministic, i.e. having very large weights, can occur in Neural Markov Logic Networks but, at least for Gibbs sampling, the effect is the same. Such distributions would naturally be learned in our framework on most datasets. Our solution in this model is to simply avoid learning distributions with determinism by adding noise during training. In particular, we set a parameter $\pi_n \in [0, 1]$ and, at the beginning of each training epoch, we inverted each ground atom of the input possible worlds (*True* to *False* and vice versa) with probability π_n.Moreover, this added noise prevents the model to perfectly fit training data, acting as a regularizer (Bishop, 1995).

7.1.4 Connections to Markov Logic Networks

Markov logic networks are a popular statistical relational framework. It turns out that every Markov logic network can be represented as a Neural Markov Logic Network with a *single* carefully selected potential function.

Kuželka et al. (2018) studies two maximum-entropy models, Model A, which is close to the model that we study in this Chapter, and Model B, which is the same as Markov logic networks. Syntactically, both models are encoded as sets of weighted first order logic formulas, e.g. $\Phi = \{(\alpha_1, w_1), \ldots, (\alpha_M, w_m)\}$. In particular, given a positive integer k, Model A defines the following distribution:

$$p_A(\omega) = \frac{1}{Z} \exp\left(\sum_{(\alpha,w)\in\Phi_A} w \cdot \#_k(\alpha, \omega) \right)$$

where Z is the normalization constant and $\#(\alpha, \omega)$ is the fraction of size-k subsets $\mathcal{S}$ of constants in the possible world ω for which $\omega\langle\mathcal{S}\rangle \models \alpha$ (i.e. the formula α is classically true in the fragment of ω induced by $\mathcal{S}$). Let us first define

$$\phi_{\alpha,w}(\gamma) = \begin{cases} w & \gamma \models \alpha \\ 0 & \gamma \not\models \alpha \end{cases}$$

It is then easy to see that the distribution $p_A(\omega)$ can also easily be encoded as a Neural Markov Logic Network by selecting the potential function $\phi(\gamma) = \sum_{(\alpha,w)\in\Phi_A} \phi_{\alpha,w}(\gamma)$ and by carefully selecting the weights β_i in the Neural Markov Logic Network.

Next we show that all distributions in Model B can be translated to distributions in Model A. First we will assume that the formulas α_i do not contain any constants.

Model B is given by

$$p_B(\omega) = \frac{1}{Z} \exp\left(\sum_{(\beta,v)\in\Phi_B} v \cdot n(\beta, \omega) \right)$$

where $n(\beta, \omega)$ is the number[1] of true injective groundings of the formula β in the possible world ω. Hence, Model B is exactly the same as Markov logic networks up to the requirement on injectivity of the groundings. However, as shown in (Buchman and Poole, 2015), any Markov logic network can be translated into such modified Markov logic network with the injectivity requirement on the groundings.

Let k be an integer greater or equal to the number of variables in any formula in Φ_B. Now, let Γ be the set of all size-k fragments. For every formula β in Φ_B, we introduce a partition $\mathcal{P}$ on Γ induced by the equivalence relation $\sim_\beta$ defined by: $\gamma \sim_\beta \gamma'$ iff $n(\beta, \gamma) = n(\beta, \gamma')$. Since β is assumed to not contain any constants, we can capture each of these equivalence classes C by a (possibly quite big) first-order logic sentence without constants β_C. Let C_i be the equivalence class that contains

[1] In (Kuželka et al., 2018), Model B is defined using *fractions* of true grounding substitutions instead of *numbers* of true grounding substitutions. However, these two definitions are equivalent up to normalizations and both work for our purposes but the latter one is a bit more convenient here. Hence we choose the latter one here.

fragments γ such that $n(\beta, \gamma) = i$. Let $m(\beta, \omega) = \sum_{C_i \in \mathcal{P}} \sum_{\gamma \in \Gamma_k(\omega)} i \cdot \mathbb{1}(\gamma \models \beta_C)$. By construction, it holds $m(\beta, \omega) = \sum_{\gamma \in \Gamma_k(\omega)} n(\beta, \gamma)$. Every true injective grounding of the formula β, having l variables, is contained in $\binom{n-l}{k-l}$ different size-k fragments of ω, each of which gives rise to $k!$ anonymized fragments in the multi-set $\Gamma_k(\omega)$. So $m(\beta, \omega)$ is over-counting the number of true groundings $n(\beta, \omega)$ by a constant factor. It follows that, by carefully selecting the weights of the formulas β_C we can encode the distribution $p_B(\omega)$ also in Model A. Although this particular transformation that we have just sketched is not very efficient, it does show that Neural Markov Logic Networks with potential functions of width k can express all distributions that can be expressed by Markov logic networks containing formulas with at most k variables.

First-order logic formulas defining Markov logic networks may also contain constants. In Neural Markov Logic Networks we may represent constants using vector-space embeddings as described in the main text. One can then easily extend the argument sketched above to the case covering Markov logic networks with constants.

7.2 Experiments

7.2.1 Implementation Details

We implemented Neural Markov Logic Networks in Tensorflow. In order to maximally exploit the parallel computations capabilities of GPUs, multiple Markov chains are run in parallel. This is also useful because expected values of gradients (see Section) are computed on uncorrelated samples, while sequential samples sampled from a unique chain are known to be highly correlated.

In experiments, the different global neural potentials Φ_i can rely on fragments of different sizes k so that for small k, the model can focus on learning very local statistics of the data, while, for large k, the model can focus on learning statistics on larger substructures. For example, if we represent molecules as a relational structure (see Section 7.2.3), rings are inherently global statistics which cannot be captured by local properties. This example underlines the importance of the choice of k for a correct modeling of the data distribution. However, since a single evaluation of $\Phi_i(w)$ requires a summation over $d = \binom{n}{k} k!$ number of terms, the number of elements of the sum grows exponentially with k (and polynomially, but very fast, with n). So exploiting large k is usually admissible only for small domain sizes n.

7.2.2 Knowledge Base Completion

In Knowledge Base Completion (KBC), we are provided with an incomplete Knowledge Base (KB) and asked to complete the missing part.

The KBC task is inherently in the transductive setting, since all the constants are exploited both during the training and testing phase. Moreover, data are provided in a positive-only fashion: we only know what is true and we cannot distinguish between unknown and false facts. Kuželka and Davis (2019) studied KBC tasks under the missing-completely-at-random assumption and showed consistency of learning by maximum-likelihood where both missing and false facts are treated in the same way as *false*. Hence, here we also provide both unknown and false facts as false facts during the training procedure.

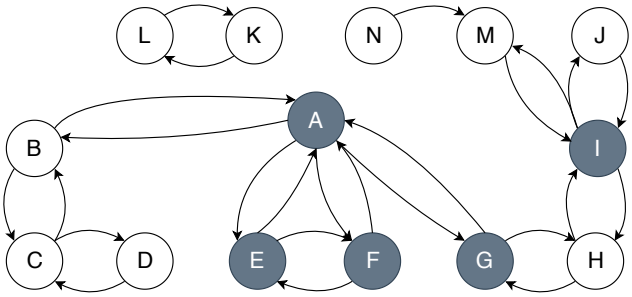
Smokers. The "Smokers" dataset (Richardson and Domingos, 2006) is a classical example in statistical relational learning literature. Here, two relations are defined on a set of constants representing people: the unary predicate *Smokes* identifies those people who smoke, while the binary predicate *friendOf* maps people to their friend. This dataset is often used to show how a statistical relational learning algorithm can model a distribution by finding a correlation of smoking habits of friends. For example, in MLNs, one typically uses weighted logical rules such as:

$$\forall x \, \forall y \, \texttt{friendOf}(x, y) \rightarrow \texttt{smokes}(x) \leftrightarrow \texttt{smokes}(y)$$

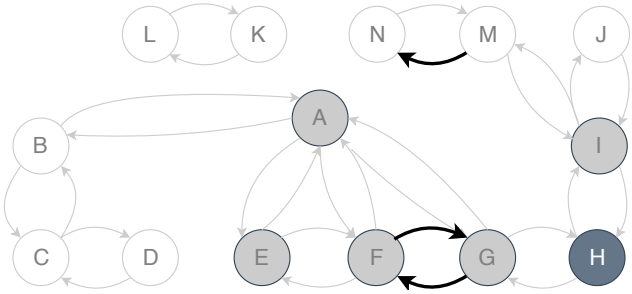
We learned a NMLN on the small smokers dataset. Since no prior knowledge about the type of rules that are relevant was used by NMLNs, the model itself had to identify which statistics are mostly informative of the provided data by learning the neural potential functions.

Here we use the Smokers dataset to define a Knowledge Base Completion task and to provide some basic intuitions about what kind of rules the model could have learned. In Figure 7.1, we show the setting before and after completion. In Figure 7.1b, we highlight only new facts whose marginal probability after training is significantly higher than the others, even though other facts has probabilities higher than the prior.

Nations. The Nations dataset (Kok and Domingos, 2007) provides information about properties and relations among countries as ground facts, like `economicaid(usa, israel)` or `embassy(israel,poland)`. There are $n = 14$ constants (i.e. nations), 56 relations and 2565 true facts. This dataset has been recently exploited for a KBC task by Rocktäschel and Riedel (2017), where some facts were removed from the dataset and the task was to predict them. The authors compared the performances of the state-of-the-art ComplEx neural model (Trouillon et al., 2017) with their proposed differentiable end-to-end neural theorem prover, showing that the combination of the two was able to outperform both of the models. Unary predicates were removed since the ComplEx model cannot deal with them. In this section, we show how we can use NMLNs to tackle a KBC task on the Nations dataset.



(a) The training KB.



(b) The completed KB.

Figure 7.1: **Knowledge Base Completion in the Nations dataset.** Circles represent constants. A grey circle means that the predicate `smokes` is *True*. A white circle means that the value of the predicate `smokes` is unknown. Links represent the relation `friendOf` (absence of an arrow means that the relation is *False*). The given world is shown on the top (7.1a), while the completed knowledge base is shown on the bottom (7.1b). The system learnt the symmetric nature of the friendship relation. It learnt that a friend of at least two smokers is also a smoker, and that two smokers, who are friends of the same person, are also friends.

We implemented the fragmented neural potentials $\phi(\gamma)$ as 2 hidden-layer neural networks, with sigmoidal hidden activations and linear output layer. The selection of the hyperparameters and the early-stopping epoch have been selected by means of a held-out validation set (the splits are same as the ones in Rocktäschel and Riedel (2017)). The size of layers has been selected from the interval $[75, 100, 150]$ for the first layer and $[30, 50, 100]$ for the second layer. The embedding size has been selected from the interval $[2, 3, 5, 10]$. The noise probability π_n has been selected from the interval $[0, 0.01, 0.02, 0.03]$. The number of parallel chains has been selected from the interval $[10, 20, 30]$.

We followed the evaluation procedure in Rocktäschel and Riedel (2017). In particular, we took a test fact and corrupted its first and second argument in all possible ways such that the corrupted fact is not in the original KB. Subsequently, we predicted a ranking of every test fact and its corruptions to calculate MRR and HITS@m. The ranking is based on marginal probabilities estimated by running Gibbs sampling on the Neural Markov Logic Network; while training the network, we also run a parallel

Table 7.1: MRR and HITS@m on Nations.

Metric	Model				
	ComplEx	**NTP**	**NTP**λ	**NMLN**	**NMLN**-Emb
MRR	0.75	0.75	0.74	0.77	**0.81**
HITS@1	0.62	0.62	0.59	0.64	**0.71**
HITS@3	0.84	0.86	**0.89**	0.86	**0.89**
HITS@10	**0.99**	**0.99**	**0.99**	**0.99**	**0.99**

Gibbs sampling chain on a state in which we fix the known part of the KB as true. Here, we compare the *ComplEx* model, the plain Neural Theorem Prover (*NTP*), the composition of the previous two (*NTP*λ), our plain model (*NMLN*) and our model when using potentials with embeddings (*NMLN-Emb*). In Table 7.1 we report the results of the KBC task on Nations. Both our models outperform competitors on the HITS@1 metric, with *NMLN-Emb* having a large gap over all the other models. It is interesting to note that the plain *NMLN* still performs better than differentiable provers, even if it is the only model which cannot exploit embeddings to perform reasoning and that has to rely only on the relational structure of fragments to make predictions. Finally, NMLN-Emb performs equally to or better than all the competitors in all the other metrics.

7.2.3 Graph generation

One of the main features differentiating our model from standard MLNs is that we learn the statistics $\phi(\gamma)$ in a differentiable manner. The obtained probability distribution is then often far more fine grained than using predefined or hand-made statistics, that are limited to what the user considers important and do not search for other interesting regularities in the data. This opens the doors to the application of NMLNs to generative tasks in non-euclidean settings, which are receiving an increasing interest recently (You et al., 2018; Li et al., 2018).

In generation tasks, our model is asked to learn the probability distribution of the relational structures induced by a graph. Indeed, any FOL-description can be considered a multi-hyper graph; thus generating in the FOL setting is applicable to generating in any graph domain. In particular, to generate graphs, we can just use the same sampling technique used during training (i.e. Gibbs Sampling) to extract new samples.

In this section, we describe a molecule generation task. We used as training data the ChEMBL molecule database (Gaulton et al., 2016). We restricted the dataset to molecules with 8 heavy atoms. We used the RDKit framework [2] to get a FOL

[2] `https://rdkit.org/`

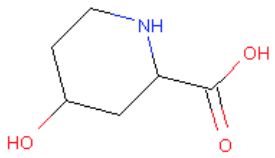


Figure 7.2: An example of molecule

representation of the molecules from their SMILES encoding.

Even though molecules can be described with a high level of precision, using both spatial features (i.e. atoms distances, bond length etc.) and chemical features (i.e. atom charge, atom mass, hybridization), in this work, we focused mainly on structural symbolic descriptions of molecules.

In particular, we described a molecule using two sets of FOL predicates:

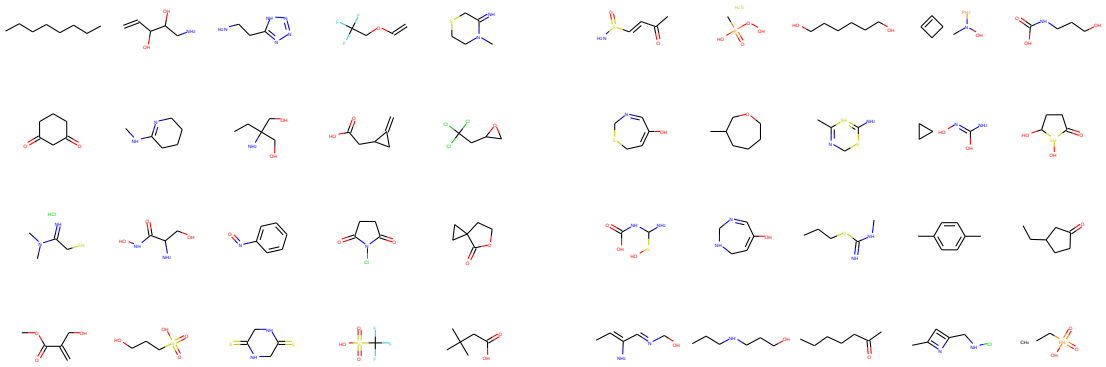
- *Atom-type unary predicates*: these are `C`, `N`, `O`, `S`, `Cl`, `F`, `P`.
- *Bond-type binary predicate*: these are `SINGLE` and `DOUBLE`.

An example of a molecule FOL description can be:

```
O(0), C(1), C(2), C(3), N(4), C(5), C(6), C(7), O(8), O(9)
SINGLE(0,1), SINGLE(1,0), SINGLE(1,2), SINGLE(2,1), SINGLE(2,3)
SINGLE(3,2), SINGLE(3,4), SINGLE(4,3), SINGLE(4,5), SINGLE(5,4)
SINGLE(5,6), SINGLE(6,5), SINGLE(5,7), SINGLE(7,5), DOUBLE(7,8)
DOUBLE(8,7), SINGLE(7,9), SINGLE(9,7), SINGLE(6,1), SINGLE(1,6)
```

We implemented the fragmented neural potentials $\phi(\gamma)$ as neural networks with sigmoidal hidden activations and linear output layer. The hyperparameters were selected from the following ranges: the number of layers in $[1, 2]$; the hidden sizes of the layers in $[30, 100, 150, 200]$; the number of fragmented potentials in $[1, 2]$, the size k of potentials in $[2, 3, 4, 5, 6]$. The number of parallel chains was set to 5.

In Figure 7.3, we show a comparison between a sample of training data and a (random) sample of molecules generated by the proposed model. In particular, 20 generated samples are chosen randomly from the last 1000 samples extracted during the training procedure. By choosing them randomly, we avoided to have very correlated samples, which is inherent in the Gibbs sampling procedure. The generated samples resembles training data both in structural patterns and variety fairly well. Furthermore, in Figure 7.4, we compare the statistics, used in Li et al.



(a) Molecules from the training data. (b) Generated molecules.

Figure 7.3: **Molecules generation.** A comparison between a sample of training data and a (random) sample of molecules generated by the proposed model. The generated samples fairly resembles training data both in structural patterns and variety. Better viewed in color.

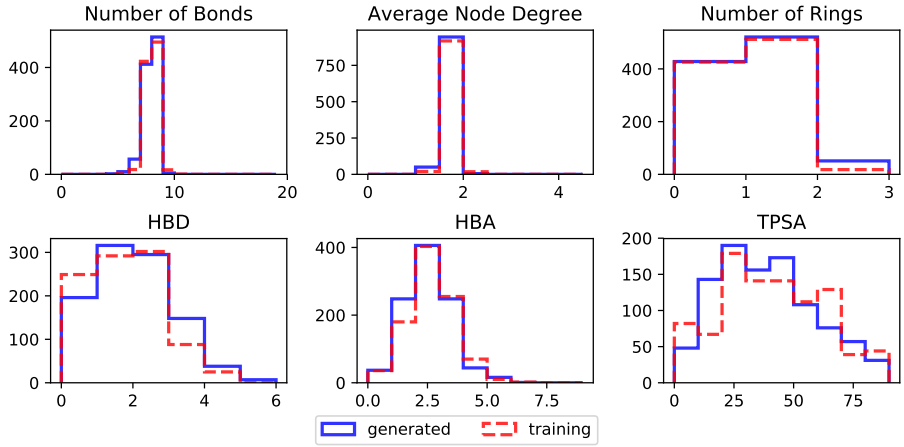


Figure 7.4: **Molecules generation.** Comparing the distributions of some chemical properties of the training data with the ones of generated data. The generated samples are capable of perfectly fitting structural properties and to very well resembling functional properties.

(2018) for a similar task, on a sample of 1000 training and generated molecules. These statistics represent both general structural properties applicable to any graph as well as chemical structural properties of molecules (e.g. the topological polar surface area (TPSA) is a topological indicator of the capability of a molecule to permeate a membrane as a function of the number of polar atoms it contains). These statistics were computed using the RDkit framework.

7.3 Conclusions

NMLNs are a MiniMax Entropy model, where we use trainable neural potentials and Gibbs Sampling inference.

The main contributions of this Chapter are:

- NMLNs are one of the few models where we can learn efficiently the structure (i.e. the potentials) of the distribution.
- NMLNs allow for the joint distribution to be learned.
- The obtained distribution is far more precise than those with hand-made potentials.
- This accuracy is showed in link prediction tasks, where NMLNs outperform Neural Theorem Provers and ComplEx.
- The precision opens the door to relational generative tasks, like molecule generation, where samples from NMLN are showed to match true data on several structural and chemical properties

However, there are still room for improvements. Indeed, in NMLNs:

- Even though being applicable in settings where exact inference is not allowed, this model is still applicable only in small domains (i.e. hundreds of constants); thus it suffers from scalability issues;
- Learning potentials as neural networks does not allow a direct interpretation of learned rules. Still, we can apply methods for interpreting neural networks.

For a certain perspective, DLMs and NMLNs represent two complementary approaches and MiniMax Entropy represent the common field allowing, in the future, their integration.

Chapter 8

Conclusions and Future Works

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 Conclusions

MiniMax Entropy Models In this thesis we presented MiniMax Entropy principle together with the correspondent optimization problem. The solution to this problem gives raise to a broad class of relational models, called MiniMax Entropy models. We have shown as this models recover classical settings in corner cases but they encompass all the intermediate variations. We showed that when conditioning MiniMax Entropy models on available perceptual data about symbolic entities, these models represent a candidate solution for a unifying theory of integration of symbolic and subsymboli reasoning. MiniMax Entropy models still remain a too general theory and they need a much better investigation in different instantiations, as the ones presented in this thesis.

LYRICS This thesis also presented a novel and general framework, called LYRICS, to bridge logic reasoning and deep learning. Its connection with MiniMax Entropy models comes when interpreting inference in neural networks as providing the one most probable explanation of some evidence on a probability distribution. This interpretation nicely recovers the learning from constraints paradigm when features are interpreted as constraints and regularization is interpreted as priors on the probability distribution. 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. A nice property of the approach is that it allows the definition of generative (adversarial) problems using the simple language of logic, allowing us to tackle very different translation problems with little effort.

Deep Logic Models. Deep Logic Models shows as MAP inference coupled with fuzzy logic potentials instantiate a MiniMax Entropy models capable of performing multiple neuro-symbolic tasks but still keeping tractability. Fuzzy t-norm theory represents a fundamental choice to make potentials differentiable w.r.t. states, allowing the exploitation of variational methods for inference. Moreover, logic potentials are shown to be a particular instance of a more general class of potentials working on fragments on the whole relational structure.

Neural Markov Logic Network. Neural Markov Logic Network fully exploits the potentiality of the Minimization of Entropy principle to learn the structure of the probability distribution under investigation. Indeed, Neural Markov Logic Networks are MiniMax Entropy models where parametric potential are defined on the entire relational structure. This allows features of the probability distribution to be learned from data and not provided by the user as in the inspiring Markov Logic Networks. By still beign able to reach state-of-the-art in link prediction, these models show that MiniMax Entropy models open the doors to generative tasks in the relational settings.

8.2 Future Works

MiniMax Entropy models are an extremely large class of models. The general algorithm for learning parameters of this model is extremely general and it can be instantiated into multiple more specific models. It simply encompasses the idea that learning can be guided by looking for explainations that are higly descriptive of the data we are observing but at the same time less biased towards anything that we do not observe.

Balancing exact and approximate inference MiniMax Entropy models are a first step towards a more general theory of relational reasoning and, in perspective, of artificial intelligence itself. However, from a foundational viewpoint, it has to be much more deeply investigated which is and how to manage *the balance between approximated and exact reasoning*. Indeed, MiniMax Entropy models allow any inference algorithm on the corresponding probability distribution to be exploited. Even though this allows a very broad tuning of the algorithms to the specific needs of a user, it still provides no insight on how we can reach the great balance that humans

show in handling the intervention of intuitive or conscious reasoning processes. We believe that a unique inference scheme (i.e. algorithm) should exist which is the superposition of an approximated scheme (e.g. a neural approximation) and an exact scheme (e.g. a search algorithm in the state space).

We are currently working on an inference scheme where exact processes delegate approximated schemes in all the cases they can not exactly carry on the inference process. This is motivated by the need for a trade-off between exact reasoning and scalability. In particular, we are looking a for a unique scheme which gradually pass from an exact computation to an approximate one when the exact computation is not feasible, and we would like that the degree of approximation to be proportional to the degree of infeasibility

Variational Inference and Hybrid MCMC Inference Another line of research that we are considering is to recur to more advanced variational methods for inference. Indeed, MAP inference can be considered the simplest case of inference and it is clearly very limited when the distribution under investigation does not meet the unimodality feature discussed in Chapter 3. However, MiniMax Entropy models with differentiable features potentially allow us to investigate any variational inference scheme.

Variational Inference techniques could also represent a valid candidate for building proposal distribution in a more general MCMC scheme, e.g. Metropolis Hastings.

Graph Neural Networks. Graph Neural Networks (GNN) represent nowadays an alternative to SRL methods to deal with relational data. Even though, they are very powerful in learning diffusion mechanisms on large networks, computing complex features on relational structures, they still struggle in modeling uncertainty over the structure. An intuitive reason behind this is that they consider the structure (i.e. edges) of the graph as an input of the algorithm. On the contrary, MiniMax Entropy schemes explicitly model uncertainty over relations. However, GNNs could find a very interesting place inside MiniMax Entropy distributions, as a powerful and efficient way to compute statistics on fragments. In fact, NMLNs do learn statistics on fragments represented as boolean vectors (i.e. a given portion of the Herbrand Interpretation). This is indeed a flat representation that is not able to exploit its relational nature. On the contrary, GNNs encoding mechanism can allow for an efficient weight sharing scheme to be exploited in potential learning.

Appendix A

Publications

Journal papers

1. *Information diffusion in a multi-social-network scenario: framework and ASP-based analysis* - G Marra, D Ursino, F Ricca, G Terracina - Knowledge and Information Systems 48, 2016 - **Candidate's contributions**: formulation of the problem, formulation of the solution and design of the experimental campaign.

2. M Maggini, **G Marra,** S Melacci, A Zugarini *Learning in Text Streams: Discovery and Disambiguation of Entity and Relation Instances* - Transaction of Neural Networks and Learning Systems, (2020, to appear) - **Candidate's contributions**: joint definition of the method, design and implementation of the agent, joint design and implementation of the experiments.

Peer reviewed conference papers

1. **G Marra,** F Giannini, M Diligenti, M Gori, M Maggini *Relational Neural Machines* - European Conference of Artificial Intelligence (ECAI 2020) **Candidate's contributions**: joint definition of the method, design and implementation of both the framework and the experiments.

2. M Tiezzi, **G Marra,** M Maggini, M Gori *Lagrangian Propagation Graph Neural Networks* - European Conference of Artificial Intelligence (ECAI 2020) **Candidate's contributions**: joint definition of the method, joint design of the experiments.

3. **G Marra** F Giannini, M Diligenti, M Gori *Integrating Learning and Reasoning with Deep Logic Models* - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD, Wurzburg, Germany2019- **Candidate's contributions**: carried out inspiring studies, joint definition of the theory, design and implementations of the experiments.

4. **G Marra**, F Giannini, M Diligenti, M Gori - *LYRICS: a General Interface Layer to Integrate AI and Deep Learning* - European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML PKDD, -Wurzburg, Germany 2019 - **Candidate's contributions**: design, development and testing of the overall framework, design and implementations of the experiments.

5. F Giannini, **G Marra** M Diligenti, M Maggini, M Gori *On the relation between Loss Functions and T-Norms* - International Conference on Inductive Logic Programming, ILP, 2019 - **Candidate's contributions**: design and development of the experiments.

6. **G Marra**, D Zanca, A Betti, M Gori - *Learning Neuron Non-Linearities with Kernel-Based Deep Neural Networks* - International Conference Of The Italian Association for Artificial Intelligence, AI*IA 2019 - **Candidate's contributions**: design and development of the experiments.

7. **G Marra** F Giannini, M Diligenti, M Gori *Constraint-Based Visual Generation* - International Conference on Artificial Neural Networks, ICANN, 2019 - **Candidate's contributions**: definition of the method, design and development of the experiments.

8. **G Marra**, A Zugarini, S Melacci, M Maggini - *An Unsupervised Character-Aware Neural Approach to Word and Context Representation Learning* - International Conference on Artificial Neural Networks, ICANN, 2018 - **Candidate's contributions**: formulation of the problem, formulation of the solution, design and development of the experimental campaign.

9. A Betti, M Gori, **G Marra** - *A Constrained-Based Approach to Machine Learning* - 14th International Conference on Signal-Image Technology & Internet, SIT, 2018 **Candidate's contributions**: implementation of proof-of-concepts experiments.

10. **G Marra**, A Nocera, F Ricca, G Terracina, D Ursino, *Investigating Node Influence Maximization and Influential Node Characterization in a Multi-Social-Network Scenario via Disjunctive Logic Programming.* - SEBD, 2014 - **Candidate's contributions**: formulation of the problem, formulation of the solution and design of the experimental campaign.

11. D Leggio, **G Marra,** D Ursino, *Defining and investigating the scope of users and hashtags in Twitter* - OTM Confederated International Conferences, 2014 - **Candidate's contributions**: formulation of the problem, supporting experimental evaluation.

12. **G Marra,** A Nocera, F Ricca, G Terracina, D Ursino - *Investigating information diffusion in a multi-social-network scenario via answer set programming* - International Conference on Web Reasoning and Rule Systems, 2014 - **Candidate's contributions**: formulation of the problem, formulation of the solution and design of the experimental campaign.

Workshop papers

1. **G Marra**, O Kuželka *Neural Markov Logic Networks* - (NeurIPS Workshop KR2ML, 2019) - **Candidate's contributions**: joint definition of theory, desing and implementation of the framework and of the experiments.

2. **G Marra,** F Ricca, G Terracina, D Ursino *Exploiting answer set programming for handling information diffusion in a multi-social-network scenario* - - European Workshop on Logics in Artificial Intelligence, 2014 - **Candidate's contributions**: formulation of the problem, formulation of the solution, design of the experimental campaign.

3. M Tiezzi, **G Marra,** M Maggini, M Gori *Lagrangian Propagation Graph Neural Networks* - Deep Learning on Graphs: Methodologies and Applications (DGLMA- AAAI 2020) **Candidate's contributions**: joint definition of the method, joint design of the experiments.

Papers under review

1. M Tiezzi, **G Marra,** S Melacci, M Maggini *Deep Constraint-based Propagation in Graph Neural Networks* - (Submitted at TPAMI - *Under Review*) **Candidate's contributions**: joint definition of the method, joint design of the experiments.

2. **G Marra,** F Giannini, M Diligenti, M Maggini, M Gori *Learning and T-Norms Theory* - **Candidate's contributions**: desing and implementation of the framework and of the experiments.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Ackley, D. H., Hinton, G. E., and Sejnowski, T. J. (1985). A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169.

Bach, S. H., Broecheler, M., Huang, B., and Getoor, L. (2017). Hinge-loss markov random fields and probabilistic soft logic. *Journal of Machine Learning Research*, 18:1–67.

Bach, S. H., Huang, B., London, B., and Getoor, L. (2013). Hinge-loss markov random fields: convex inference for structured prediction. In *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, pages 32–41. AUAI Press.

Bishop, C. M. (1995). Training with noise is equivalent to tikhonov regularization. *Neural computation*, 7(1):108–116.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013). Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795.

Bouchard, G., Singh, S., and Trouillon, T. (2015). On approximate reasoning capabilities of low-rank vector spaces. *AAAI Spring Syposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches*.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

Buchman, D. and Poole, D. (2015). Representing aggregators in relational probabilistic models. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

Chang, J. and Blei, D. (2009). Relational topic models for document networks. In *Artificial Intelligence and Statistics*, pages 81–88.

Choi, J. and Amir, E. (2012). Lifted relational variational inference. *arXiv preprint arXiv:1210.4867*.

Cohen, W. W. (2016). Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Cover, T. M. and Thomas, J. A. (2012). *Elements of information theory*. John Wiley & Sons.

Czarnecki, W. M., Osindero, S., Jaderberg, M., Swirszcz, G., and Pascanu, R. (2017). Sobolev training for neural networks. In *Advances in Neural Information Processing Systems*, pages 4278–4287.

De Raedt, L. and Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*, pages 1–27. Springer.

De Raedt, L., Manhaeve, R., Dumancic, S., Demeester, T., and Kimmig, A. (2019). Neuro-symbolic= neural+ logical+ probabilistic. In *NeSy'19@ IJCAI, the 14th International Workshop on Neural-Symbolic Learning and Reasoning*, pages 1–4.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.

Diligenti, M., Gori, M., and Sacca, C. (2017). Semantic-based regularization for learning and inference. *Artificial Intelligence*, 244:143–165.

Du, Y. and Mordatch, I. (2019). Implicit generation and generalization in energy-based models. *arXiv preprint arXiv:1903.08689*.

Fakhraei, S., Foulds, J., Shashanka, M., and Getoor, L. (2015). Collective spammer detection in evolving multi-relational social networks. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1769–1778. ACM.

Garcez, A. S. d., Broda, K. B., and Gabbay, D. M. (2012). *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media.

Gaulton, A., Hersey, A., Nowotka, M., Bento, A. P., Chambers, J., Mendez, D., Mutowo, P., Atkinson, F., Bellis, L. J., Cibrián-Uhalte, E., et al. (2016). The chembl database in 2017. *Nucleic acids research*, 45(D1):D945–D954.

Giannini, F., Diligenti, M., Gori, M., and Maggini, M. (2018). On a convex logic fragment for learning and reasoning. *IEEE Transactions on Fuzzy Systems*.

Giannini, F., Marra, G., Diligenti, M., Maggini, M., and Gori, M. (2019). On the relation between loss functions and t-norms. *Conference in Inductive Logic Programming*.

Gnecco, G., Gori, M., Melacci, S., and Sanguineti, M. (2015). Foundations of support constraint machines. *Neural computation*, 27(2):388–480.

Goodfellow, I., Bengio, Y., Courville, A., and Bengio, Y. (2016). *Deep learning*, volume 1. MIT press Cambridge.

Gori, M. (2017). *Machine Learning: A constraint-based approach*. Morgan Kaufmann.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Gutiérrez-Basulto, V., Jung, J. C., and Kuzelka, O. (2018). Quantified markov logic networks. In *KR*.

Hájek, P. (2013). *Metamathematics of fuzzy logic*, volume 4. Springer Science & Business Media.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800.

Huang, Z., Xu, W., and Yu, K. (2015). Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Jaynes, E. T. (1957a). Information theory and statistical mechanics. *Physical review*, 106(4):620.

Jaynes, E. T. (1957b). Information theory and statistical mechanics. ii. *Physical review*, 108(2):171.

Jenei, S. (2002). A note on the ordinal sum theorem and its consequence for the construction of triangular norms. *Fuzzy Sets and Systems*, 126(2):199–205.

Kahneman, D. (2011). *Thinking, fast and slow*. Macmillan.

Kaipio, J. and Somersalo, E. (2006). *Statistical and computational inverse problems*, volume 160. Springer Science & Business Media.

Kern-Isberner, G., Wilhelm, M., and Beierle, C. (2017). Probabilistic knowledge representation using the principle of maximum entropy and gröbner basis theory. *Annals of Mathematics and Artificial Intelligence*, 79(1-3):163–179.

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python*, pages 195–208. Springer.

Kimmig, A., Van den Broeck, G., and De Raedt, L. (2011). An algebraic prolog for reasoning about possible worlds. In *AAAI*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Klement, E., P., Mesiar, R., and Pap, E. (2004a). Triangular norms. position paper i: basic analytical and algebraic properties. *Fuzzy Sets and Systems*, 143(1):5–26.

Klement, E., P., Mesiar, R., and Pap, E. (2004b). Triangular norms. position paper ii: general constructions and parameterized families. *Fuzzy Sets and Systems*, 145(3):411–438.

Klement, E., P., Mesiar, R., and Pap, E. (2004c). Triangular norms. position paper iii: Continuous t-norms. *Fuzzy Sets and Systems*, 145:439–454.

Klement, E., P., Mesiar, R., and Pap, E. (2013). *Triangular norms*, volume 8. Springer Science & Business Media.

Kok, S. and Domingos, P. (2005). Learning the structure of markov logic networks. In *Proceedings of the 22nd international conference on Machine learning*, pages 441–448. ACM.

Kok, S. and Domingos, P. (2007). Statistical predicate invention. In *Proceedings of the 24th international conference on Machine learning*, pages 433–440. ACM.

Kolb, S., Teso, S., Passerini, A., and De Raedt, L. (2018). Learning smt (lra) constraints using smt solvers. In *IJCAI*, pages 2333–2340.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Kuželka, O. and Davis, J. (2019). Markov logic networks for knowledge base completion: A theoretical analysis under the MCAR assumption. In *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019 (to appear)*.

Kuželka, O., Wang, Y., Davis, J., and Schockaert, S. (2018). Relational marginal problems: Theory and estimation. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Landwehr, N., Passerini, A., De Raedt, L., and Frasconi, P. (2010). Fast learning of relational kernels. *Machine learning*, 78(3):305–342.

Li, C., Liu, H., Chen, C., Pu, Y., Chen, L., Henao, R., and Carin, L. (2017). Alice: Towards understanding adversarial learning for joint distribution matching. In *Advances in Neural Information Processing Systems*, pages 5501–5509.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. (2018). Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*.

Liang, P., Daumé III, H., and Klein, D. (2008). Structure compilation: trading structure for features. In *ICML*, pages 592–599.

Lippi, M. and Frasconi, P. (2009). Prediction of protein β-residue contacts by markov logic networks with grounding-specific weights. *Bioinformatics*, 25(18):2326–2333.

Liu, M.-Y., Breuel, T., and Kautz, J. (2017). Unsupervised image-to-image translation networks. In *Advances in Neural Information Processing Systems*, pages 700–708.

Liu, M.-Y. and Tuzel, O. (2016). Coupled generative adversarial networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 469–477. Curran Associates Inc.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Lu, Q. and Getoor, L. (2003). Link-based classification. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pages 496–503.

Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., and De Raedt, L. (2018). Deepproblog: Neural probabilistic logic programming. *arXiv preprint arXiv:1805.10872*.

Marra, G., Giannini, F., Diligenti, M., and Gori, M. (2019a). Integrating learning and reasoning with deep logic models. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML- PKDD*.

Marra, G., Giannini, F., Diligenti, M., and Gori, M. (2019b). Lyrics: a general interface layer to integrate ai and deep learning. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, ECML- PKDD*.

Marra, G., Giannini, F., Diligenti, M., Maggini, M., and Gori, M. (2019c). Learning and t-norms theory. *arXiv preprint arXiv:1907.11468*.

Marra, G. and Kuželka, O. (2019). Neural markov logic networks. *arXiv preprint arXiv:1905.13462, to be presented at Knowledge Representation & Reasoning Meets Machine Learning Workshop at NeurIPS 2019*.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Minervini, P., Bosnjak, M., Rocktäschel, T., and Riedel, S. (2018). Towards neural theorem proving at scale. *arXiv preprint arXiv:1807.08204*.

Mnih, A. and Kavukcuoglu, K. (2013). Learning word embeddings efficiently with noise-contrastive estimation. In *Advances in neural information processing systems*, pages 2265–2273.

Mostert, P. S. and Shields, A. L. (1957). On the structure of semigroups on a compact manifold with boundary. *Annals of Mathematics*, pages 117–143.

Muggleton, S. (1991). Inductive logic programming. *New generation computing*, 8(4):295–318.

Muggleton, S. (1994). Bayesian inductive logic programming. In *Proceedings of the seventh annual conference on Computational learning theory*, pages 3–11. ACM.

Muggleton, S. and De Raedt, L. (1994). Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679.

Muggleton, S., Lodhi, H., Amini, A., and Sternberg, M. J. (2005). Support vector inductive logic programming. In *Discovery science*, volume 3735, pages 163–175. Springer.

Neal, R. M. and Hinton, G. E. (1998). A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer.

Neville, J. and Jensen, D. (2000). Iterative classification in relational data. In *Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, pages 13–20.

Novák, V., Perfilieva, I., and Mockor, J. (2012). *Mathematical principles of fuzzy logic*, volume 517. Springer Science & Business Media.

Poon, H. and Domingos, P. (2006). Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, volume 6, pages 458–463.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*, 62(1-2):107–136.

Robert, C. and Casella, G. (2013). *Monte Carlo statistical methods*. Springer Science & Business Media.

Rocktäschel, T. and Riedel, S. (2016). Learning knowledge base inference with neural theorem provers. In *Proceedings of the 5th Workshop on Automated Knowledge Base Construction*, pages 45–50.

Rocktäschel, T. and Riedel, S. (2017). End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pages 3788–3800.

Rosca, M., Lakshminarayanan, B., and Mohamed, D. W.-F. S. (2017). Variational approaches for auto-encoding generative adversarial networks. *stat*, 1050:15.

Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, pages 791–798. ACM.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., and Eliassi-Rad, T. (2008). Collective classification in network data. *AI magazine*, 29(3):93.

Serafini, L. and Garcez, A. d. (2016a). Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422*.

Serafini, L. and Garcez, A. S. d. (2016b). Learning and reasoning with logic tensor networks. In *AI* IA*, pages 334–348.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423.

Sourek, G., Aschenbrenner, V., Zelezny, F., and Kuzelka, O. (2015). Lifted relational neural networks. *arXiv preprint arXiv:1508.05128*.

Sourek, G., Aschenbrenner, V., Železný, F., Schockaert, S., and Kuzelka, O. (2018). Lifted relational neural networks: Efficient learning of latent relational structures. *J. Artif. Intell. Res.*, 62:69–100.

Tjong Kim Sang, E. F. and Buchholz, S. (2000). Introduction to the conll-2000 shared task: Chunking. In *Proceedings of the 2nd workshop on Learning language in logic and the 4th conference on Computational natural language learning-Volume 7*, pages 127–132. Association for Computational Linguistics.

Trouillon, T., Dance, C. R., Gaussier, É., Welbl, J., Riedel, S., and Bouchard, G. (2017). Knowledge graph completion via complex tensor factorization. *The Journal of Machine Learning Research*, 18(1):4735–4772.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. (2016). Complex embeddings for simple link prediction. In *International Conference on Machine Learning*, pages 2071–2080.

Tu, L. and Gimpel, K. (2018). Learning approximate inference networks for structured prediction. *arXiv preprint arXiv:1803.03376*.

Van Gelder, A., Ross, K. A., and Schlipf, J. S. (1991). The well-founded semantics for general logic programs. *Journal of the ACM (JACM)*, 38(3):619–649.

Wang, W. Y. and Cohen, W. W. (2016). Learning first-order logic embeddings via matrix factorization. In *IJCAI*, pages 2132–2138.

Wang, W. Y., Mazaitis, K., and Cohen, W. W. (2013). Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2129–2138. ACM.

Weston, J., Chopra, S., and Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

Wiegand, T., Schwarz, H., et al. (2011). Source coding: Part i of fundamentals of source and video coding. *Foundations and Trends® in Signal Processing*, 4(1–2):1–222.

Xu, K., Hu, W., Leskovec, J., and Jegelka, S. (2018). How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. (2018). Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2223–2232.

Zhu, S. C., Wu, Y. N., and Mumford, D. (1997). Minimax entropy principle and its application to texture modeling. *Neural computation*, 9(8):1627–1660.