Integrating Learning and Reasoning with Deep Logic Models*

Giuseppe Marra[1,2], Francesco Giannini[2],
Michelangelo Diligenti[2] (✉), and Marco Gori[2]

[1] Department of Information Engineering, University of Florence, ITALY
g.marra@unifi.it
[2] Department of Information Engineering and Mathematical Sciences,
University of Siena, ITALY
{fgiannini,diligmic,marco}@diism.unisi.it

Abstract. Deep learning is very effective at jointly learning feature representations and classification models, especially when dealing with high dimensional input patterns. Probabilistic logic reasoning, on the other hand, is capable of take consistent and robust decisions in complex environments. The integration of deep learning and logic reasoning is still an open-research problem and it is considered to be the key for the development of real intelligent agents. This paper presents Deep Logic Models, which are deep graphical models integrating deep learning and logic reasoning both for learning and inference. Deep Logic Models create an end-to-end differentiable architecture, where deep learners are embedded into a network implementing a continuous relaxation of the logic knowledge. The learning process allows to jointly learn the weights of the deep learners and the meta-parameters controlling the high-level reasoning. The experimental results show that the proposed methodology overcomes the limitations of the other approaches that have been proposed to bridge deep learning and reasoning.

Keywords: ML and logic integration· probabilistic reasoning· ML and constrains

1 Introduction

Artificial Intelligence (AI) approaches can be generally divided into symbolic and sub-symbolic approaches. Sub-symbolic approaches like artificial neural networks have attracted most attention of the AI community in the last few years. Indeed, sub-symbolic approaches have got a large competitive advantage from the availability of a large amount of labeled data in some applications. In these contexts, sub-symbolic approaches and, in particular, deep learning ones are effective in processing low-level perception inputs [3,18]. For instance, deep learning architectures have achieved state-of-the-art results in a wide range of

---

tasks, e.g. speech recognition, computer vision, natural language processing, where deep learning can effectively develop feature representations and classification models at the same time.

On the other hand, symbolic reasoning [7,16,23], which is typically based on logical and probabilistic inference, allows to perform high-level reasoning (possibly under uncertainty) without having to deal with thousands of learning hyper-parameters. Even though recent work has tried to gain insight on how a deep model works [21], sub-symbolic approaches are still mostly seen as *black-boxes*, whereas symbolic approaches are generally easier to interpret, as the symbol manipulation or chain of reasoning can be unfolded to provide an understandable explanation to a human operator.

In spite of the incredible success of deep learning, many researchers have recently started to question the ability of deep learning to bring us real AI, because the amount and quality of training data would explode in order to jointly learn the high-level reasoning that is needed to perform complex tasks [2]. For example, forcing some structure to the output of a deep learner has been shown to bring benefits in image segmentation tasks, even when simple output correlations were added to the enforced contextual information [6].

Blending symbolic and sub-symbolic approaches is one of the most challenging open problem in AI and, recently, a lot of works, often referred as neuro-symbolic approaches [10], have been proposed by several authors [6,14,22,27]. In this paper, we present Deep Logic Models (DLMs), a unified framework to integrate logical reasoning and deep learning. DLMs bridge an input layer processing the sensory input patterns, like images, video, text, from a higher level which enforces some structure to the model output. Unlike in Semantic-based Regularization [8] or Logic Tensor Networks [9], the sensory and reasoning layers can be jointly trained, so that the high-level weights imposing the output structure are jointly learned together with the neural network weights, processing the low-level input. The bonding is very general as any (set of) deep learners can be integrated and any output structure can be expressed. This paper will mainly focus on expressing the high-level structure using logic formalism like first–order logic (FOL). In particular, a consistent and fully differentiable relaxation of FOL is used to map the knowledge into a set of potentials that can be used in training and inference.

The outline of the paper is the following. Section 2 presents the model and the integration of logic and learning. Section 3 compares and connects the presented work with previous work in the literature and Section 4 shows the experimental evaluation of the proposed ideas on various datasets. Finally, Section 5 draws some conclusions and highlights some planned future work.

2   Model

We indicate as $\boldsymbol{\theta}$ the model parameters, and X the collection of input sensory data. Deep Logic Models (DLMs) assume that the prediction of the system is constrained by the available prior knowledge. Therefore, unlike standard Neural networks which compute the output via a single forward pass, the output
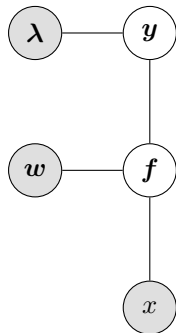
Fig. 1. The DLM graphical model assumes that the output variables $\boldsymbol{y}$ depend on the output of first stage $\boldsymbol{f}$, processing the input X. This corresponds to the breakdown into a lower sensory layer and a high level semantic one.

computation in a DLM can be decomposed into two stages: a *low-level* stage processing the input patterns, and a subsequent *semantic* stage, expressing constraints over the output and performing higher level reasoning. We indicate by $\boldsymbol{y} = \{y_1, \ldots, y_n\}$ and by $\boldsymbol{f} = \{f_1, \ldots, f_n\}$ the two multivariate random variables corresponding to the output of the model and to the output of the first stage respectively, where $n > 0$ denotes the dimension of the model outcomes. Assuming that the input data is processed using neural networks, the model parameters can be split into two independent components $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{\lambda}\}$, where $\boldsymbol{w}$ is the vector of weights of the networks f_{nn} and $\boldsymbol{\lambda}$ is the vector of weights of the second stage, controlling the semantic layer and the constraint enforcement. Figure 1 shows the graphical dependencies among the stochastic variables that are involved in our model. The first layer processes the inputs returning the values $\boldsymbol{f}$ using a model with parameters $\boldsymbol{w}$. The higher layer takes as input $\boldsymbol{f}$ and applies reasoning using a set of constraints, whose parameters are indicated as $\boldsymbol{\lambda}$, then it returns the set of output variables $\boldsymbol{y}$.

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

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

Assuming the breakdown into a sensory and a semantic level, the prior may be decomposed as $p(\boldsymbol{\theta}) = p(\boldsymbol{\lambda})p(\boldsymbol{w})$, while the posterior can be computed by marginalizing over the assignments for $\boldsymbol{f}$:

$$p(\boldsymbol{y}|\boldsymbol{\theta}, X) = \int_{\boldsymbol{f}} p(\boldsymbol{y}|\boldsymbol{f}, \boldsymbol{\lambda}) \cdot p(\boldsymbol{f}|\boldsymbol{w}, X)d\boldsymbol{f} \ . \tag{1}$$

A typical choice is to link $p(\boldsymbol{f}|\boldsymbol{w}, X)$ to the outputs of the neural architectures:

$$p(\boldsymbol{f}|\boldsymbol{w}, X) = \frac{1}{Z_{\boldsymbol{f}}} \exp\left(-\frac{(\boldsymbol{f} - \boldsymbol{f}_{nn})^2}{2\sigma^2}\right) \ ,$$

where the actual (deterministic) output of the networks f_{nn} over the inputs is indicated as $\boldsymbol{f}_{nn}$ and $Z_{\boldsymbol{f}}$ indicates the partition function with respect to $\boldsymbol{f}$. Please note that there is a one-to-one correspondence among each element of $\boldsymbol{y}, \boldsymbol{f}$ and $\boldsymbol{f}_{nn}$, such that $|\boldsymbol{y}| = |\boldsymbol{f}| = |\boldsymbol{f}_{nn}|$.

However, the integral in Equation (1) is too expensive to compute and, as commonly done in the deep learning community, only the actual output of the network is considered, namely:

$$p(\boldsymbol{f}|\boldsymbol{w}, X) \approx \delta(\boldsymbol{f} - \boldsymbol{f}_{nn}) \ ,$$

resulting in the following approximation of the posterior:

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

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

$$p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda}) \triangleq \frac{1}{Z_{\boldsymbol{y}}} \exp\left(\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \Phi_c(\boldsymbol{y}) \right) \ , \tag{2}$$

where the Φ_c are potential functions expressing some constraints on the output variables, $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_C\}$ are parameters controlling the confidence for the single constraints where a higher value corresponds to a stronger enforcement of the corresponding constraint, Φ_r is a potential that favors solutions where the output closely follows the predictions provided by the neural networks (for instance $\Phi_r(\boldsymbol{y}, \boldsymbol{f}_{nn}) = -\frac{1}{2}||\boldsymbol{y} - \boldsymbol{f}_{nn}||^2$) and $Z_{\boldsymbol{y}}$ is a normalization factor (i.e. the partition function with respect to the random variable $\boldsymbol{y}$):

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

2.1   MAP Inference

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

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

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

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

2.2   Learning

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

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

In particular, assuming that $p(\boldsymbol{y}_t|\boldsymbol{\theta}, X)$ follows the model defined in equation (2) and the parameter priors follow Gaussian distributions, yields:

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

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

$$\frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{y}_t, X)}{\partial \lambda_c} = -\beta \lambda_c + \Phi_c(\boldsymbol{y}_t) - E_p\left[\Phi_c\right]$$

$$\frac{\partial \log p(\boldsymbol{\theta}|\boldsymbol{y}_t, X)}{\partial w_i} = -\alpha w_i + \frac{\partial \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn})}{\partial w_i} - E_p\left[\frac{\partial \Phi_r}{\partial w_i}\right]$$

Unfortunately, the direct computation of the expected values in the above derivatives is not feasible. A possible approximation [12,13] relies on replacing the expected values with the corresponding value at the MAP solution, assuming that most of the probability mass of the distribution is centered around it. This can be done directly on the above expressions for the derivatives or in the log likelihood:

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

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

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: $\log p(\boldsymbol{w}) + \log p(\boldsymbol{\lambda}) + \Phi_r(\boldsymbol{y}_t, \boldsymbol{f}_{nn}) + \Phi_r(\boldsymbol{y}_M, \boldsymbol{f}_{nn}) + \sum_c \lambda_c \left[\Phi_c(\boldsymbol{y}_t) - \Phi_c(\boldsymbol{y}_M)\right]$. In this paper, a quadratic form for the priors and the potentials is selected, but other choices are possible. For example, $\Phi_r(\cdot)$ could instead be implemented as a negative cross entropy loss. Therefore, replacing the selected forms for the potentials and changing the sign to transform a

Data: Input data X, output targets $\boldsymbol{y}_t$, function models with weights $\boldsymbol{w}$
Result: Trained model parameters $\boldsymbol{\theta} = \{\boldsymbol{\lambda}, \boldsymbol{w}\}$
Initialize $i = 0$, $\boldsymbol{\lambda} = \boldsymbol{0}$, random $\boldsymbol{w}$;
while *not converged* $\wedge$ $i < max\_iterations$ **do**
    Compute function outputs $\boldsymbol{f}_{nn}$ on X using current function weights $\boldsymbol{w}$;
    Compute MAP solution $\boldsymbol{y}_M = \text{argmax}_{\boldsymbol{y}} \log p(\boldsymbol{y}|\boldsymbol{f}_{nn}, \boldsymbol{\lambda})$;
    Compute gradient $\nabla_{\boldsymbol{\theta}} C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, X)$;
    Update $\boldsymbol{\theta}$ via gradient descent: $\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \lambda_{lr} \cdot \nabla_{\boldsymbol{\theta}} C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, X)$;
    Set $i = i + 1$;
end

Algorithm 1: Iterative algorithm to train the function weights $\boldsymbol{w}$ and the constraint weights $\boldsymbol{\lambda}$.

maximization into a minimization problem, yields the following cost function, given the current MAP solution:

$$C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, X) = \frac{\alpha}{2}||\boldsymbol{w}||^2 + \frac{\beta}{2}||\boldsymbol{\lambda}||^2 + \frac{1}{2}||\boldsymbol{y}_t - \boldsymbol{f}_{nn}||^2 +$$
$$+ \frac{1}{2}||\boldsymbol{y}_M - \boldsymbol{f}_{nn}||^2 + \sum_c \lambda_c \left[\Phi_c(\boldsymbol{y}_t) - \Phi_c(\boldsymbol{y}_M) \right] \ .$$

Minimizing $C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, X)$ 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 $C_{\boldsymbol{\theta}}(\boldsymbol{y}_t, \boldsymbol{y}_M, X)$ and one weight update step as summarized by Algorithm 1. For any constraint c, the parameter λ_c admits also a negative value. This is in case the c-th constraint turns out to be also satisfied by the actual MAP solution with respect to the satisfaction degree on the training data.

2.3   Mapping Constraints into a Continuous Logic

The DLM model is absolutely general in terms of the constraints that can be expressed on the outputs. However, this paper mainly focuses on constraints expressed in the output space $\boldsymbol{y}$ by means of first–order logic formulas. Therefore, this section focuses on defining a methodology to integrate prior knowledge expressed via FOL into a continuous optimization process.

In this framework we only deal with closed FOL formulas, namely formulas where any variable occurring in predicates is quantified. In the following, given an m-ary predicate p and a tuple $(a_1, \ldots, a_m) \in Dom(p)$, we say that $p(a_1, \ldots, a_m) \in [0, 1]$ is a *grounding* of p. Given a grounding of the variables occurring in a FOL formula (namely a grounding for all the predicates involved in the formula), the truth degree of the formula for that grounding is computed using the t-norm fuzzy logic theory as proposed in [24]. The overall degree of satisfaction of a FOL formula is obtained by grounding all the variables in such formula and aggregating the values with different operators depending on the occurring quantifiers. The details of this process are explained in the following section.

t-norm / operation	Gödel	Product	Łukasiewicz
$a \wedge b$	$\min(a, b)$	$a \cdot b$	$\max(0, a + b - 1)$
$a \vee b$	$\max(a, b)$	$a + b - a \cdot b$	$\min(1, a + b)$
$\neg a$	$1 - a$	$1 - a$	$1 - a$
$a \Rightarrow b$	$a \leq b ? 1 : b$	$\min(1, \frac{b}{a})$	$\min(1, 1 - a + b)$

Table 1. The Operations performed by the single units of an expression tree depending on the inputs a, b and the used t-norm.

Grounded Expressions. Any fully grounded FOL rule corresponds to an expression in propositional logic and we start showing how a propositional logic expression may be converted into a differentiable form. In particular, one expression tree is built for each considered grounded FOL rule, where any occurrence of the basic logic connectives $(\neg, \wedge, \vee, \Rightarrow)$ is replaced by a unit computing its corresponding fuzzy logic operation according to a certain logic semantics. In this regard, some recent work shows how to get convex (or even linear) functional constraints exploiting the convex Łukasiewicz fragment [11]. The expression tree can take as input the output values of the grounded predicates and then recursively compute the output values of all the nodes in the expression tree. The value obtained on the root node is the result of the evaluation of the expression given the input grounded predicates.

Table 1 shows the algebraic operations corresponding to the logic operators for different selections of the t-norms. Please note that the logic operators are always monotonic with respect of any single variable, but they are not always differentiable (nor even continuous). However, the sub-space where the operators are non-differentiable has null-Lebesgue measure, therefore they do not pose any practical issue, when used as part of a gradient descent optimization schema as detailed in the following.

We assume that the input data X can be divided into a set of sub-domains $X = \{X_1, X_2, \ldots\}$, such that each variable v_i of a FOL formula ranges over the data of one input domain, namely $v_i \in X_{d_i}$, where d_i is the index of the domain for the variable v_i.

For example, let us consider the rule $\forall v_1 \forall v_2 \; \neg A(v_1, v_2) \wedge B(v_1)$. For any assignment to v_1 and v_2, the expression tree returns the output value $[1 - A(v_1, v_2)] \cdot B(v_1)$, assuming to exploit the product t-norm to convert the connectives.

Quantifiers. The truth degree of a formula containing an expression with a universally quantified variable v_i is computed as the average of the t-norm truth degree of the expression, when grounding v_i over its domain. The truth degree of the existential quantifier is the *maximum* of the t-norm expression grounded over the domain of the quantified variable. When multiple quantified variables are present, the conversion is performed from the outer to the inner variable. When only universal quantifiers are present the aggregation is equivalent to the overall average over each grounding.
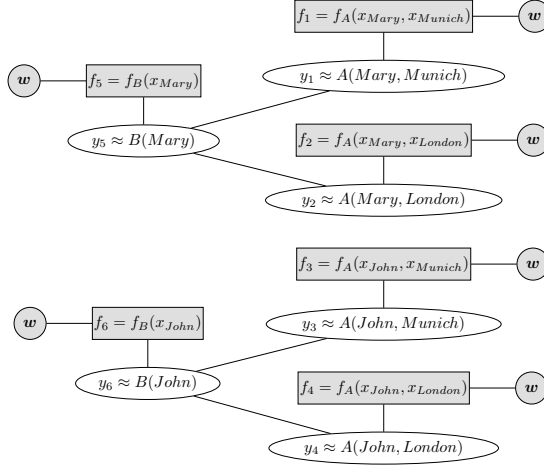
Fig. 2. The undirected graphical model built by a DLM for the rule $\forall v_1 \forall v_2 \ \neg A(v_1, v_2) \wedge B(v_1)$ where v_1 can assume values over the constants $\{Mary, John\}$ and v_2 over $\{Munich, London\}$. Each stochastic node y_i approximates one grounded predicate, while the f_i nodes are the actual output of a network getting the pattern representations of a grounding. Connections of all latent nodes y_i to the parameters $\boldsymbol{\lambda}$ have been omitted to keep the picture readable.

In the previous example, this yields the expression:

$$\Phi(X,A,B) = \frac{1}{|X_{d_1}||X_{d_2}|} \sum_{v_1 \in X_{d_1}} \sum_{v_2 \in X_{d_2}} [1 - A(v_1, v_2)] \cdot B(v_1) \ . \tag{3}$$

2.4   Potentials expressing the logic knowledge

It is now possible to explain how to build the potentials from the prior knowledge. In any learning task, each unknown grounded predicate corresponds to one variable in the vector $\boldsymbol{y}$. In the above example, the number of groundings is $|X_{d_1}| \times |X_{d_2}|$ (i.e. the size of the Cartesian product of the domains of A) and $|X_{d_1}|$ (i.e. the size of the domain of B). Assuming that both predicates A, B are unknown, $|\boldsymbol{y}| = |\boldsymbol{f}| = |X_{d_1}| \times |X_{d_2}| + |X_{d_1}|$. The vector $\boldsymbol{f}_{nn}$ is built by replacing each predicate with its NN implementation and then by considering the function outputs for each grounding in the vector. For the example:

$$\boldsymbol{f}_{nn} = \{f_A(v_{11}, v_{21}), \ldots, f_A(v_{1|X_{d_1}|}, v_{2|X_{d_2}|}), \quad f_B(v_{11}), \ldots, f_B(v_{1|d_2|})\} \ ,$$

where v_{ij} is the j-th grounding for the i-th variable and f_A, f_B are the learned neural approximations of A and B, respectively. Finally, the differentiable potential for the example formula is obtained by replacing in Equation (3) each grounded predicate with the corresponding stochastic variable in $\boldsymbol{y}$.

Figure 2 shows the undirected graphical model corresponding to the DLM for the running example rule used in this section, assuming that v_1 can assume

values over the constants $\{Mary, John\}$ and v_2 over $\{Munich, London\}$. Each stochastic node y_i approximates one grounded predicate, while the f_i nodes are the actual output of a neural network getting as input the pattern representations of the corresponding grounding. The vertical connections between two y_i and f_i nodes correspond to the cliques over the groundings for which the Φ_r potential can be decomposed. The links between the y_i nodes corresponds to the cliques over the groundings of the rule for which the corresponding Φ_c potential can be decomposed. The structure of these latter cliques follows a template determined by the rule, that is repeated for the single groundings. The graphical model is similar to the ones built by Probabilistic Soft Logic [1] or Markov Logic Networks [26], but enriched with the nodes corresponding to the output of the neural networks.

3   Related Works

DLMs have their roots in Probabilistic Soft Logic (PSL) [1], a probabilistic logic using an undirected graphical model to represent a grounded FOL knowledge base, and employing a similar differentiable approximation of FOL and allows to learn the weight of each formula in the KB by maximizing the log likelihood of the training data like done in DLMs. PSL restricts the logic that can be processed to a fragment of FOL corresponding to convex constraints. Furthermore, the rule weights are restricted to only positive values denoting how far the rule is from being satisfied. On the other hand, rule weights denote the needed constraint reactions to match the degree satisfaction of the training data in DLMs, therefore they can assume negative weights. In addition, unlike DLMs, PSL focuses on logic reasoning without any integration with deep learners, beside a simple stacking with no joint training.

The integration of learning from data and symbolic reasoning [10] has recently attracted a lot of attention. Many works in this area have emerged like Hu at al. [15], Semantic-based regularization (SBR) [8] applying these idea to kernel machines and Logic Tensor Networks (LTN) [9] which work on neural networks. All these works share the same basic idea of integrating logic reasoning and learning using a similar continuous relaxation of logic to the one presented in this paper. However, this class of approaches considers the reasoning layer as frozen, without allowing to jointly train its parameters. This is a big limitation, as these methods work better only with hard constraints, while they are less suitable in presence of reasoning under uncertainty.

DeepProbLog [22] extends the popular ProbLog [7] probabilistic programming framework with the integration of deep learners. DeepProbLog requires the output from the neural networks to be probabilities and an independence assumption among atoms in the logic is required to make inference tractable. This is a strong restriction, since the sub-symbolic layer often consists of several neural layers sharing weights.

A Neural Theorem Prover (NTP) [27,28] is an end-to-end differentiable prover based on the Prolog's backward chaining algorithm. An NTP constructs an

end-to-end differentiable architecture capable of proving queries to a KB using sub-symbolic vector representations. NTPs have been proven to be effective in tasks like entity linking and knowledge base completion. However, an NTP encodes relations as vectors using a frozen pre-selected function (like cosine similarity). This can be ineffective in modeling relations with a complex and multifaceted nature (for example a relation `friend(A,B)` can be triggered by different relationships of the representations in the embedding space). On the other hand, DLMs allow a relation to be encoded by any selected function (e.g. any deep neural networks), which is co-trained during learning. Therefore, DLMs are capable of a more powerful and flexible exploitation of the representation space. However, DLMs require to fully ground a KB (like SBR, LTN, PSL and most of other methods discussed here), while NTPs expands only the groundings on the explored frontier, which can be more efficient in some cases.

The integration of deep learning with Conditional Random Fields (CRFs) [20] is also an alternative approach to enforce some structure on the network output. This approach has been proved to be quite successful on sequence labeling for natural language processing tasks. This methodology can be seen as a special case of the more general methodology presented in this paper, when the potential functions are used to represent the correlations among consecutive outputs of a recurrent deep network.

Deep Structured Models [6,19] use a similar graphical model to bridge the sensory and semantic levels. However, they have mainly focused on imposing correlations on the output layer, without any focus on logic reasoning. Furthermore, DLMs transform the training process into an iterative constrained optimization problem, which is very different from the approximation of the partition function used in Deep Structured Models.

DLMs also open up the possibility to iteratively integrate rule induction mechanisms like the ones proposed by the Inductive Logic Programming community [17,25].

4   Experimental Results

4.1   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 compiled 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 3. A correct classification happens when both digit in a pair are correctly predicted.

Fig. 3. 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 $\forall(x, y) \; D_1(x) \to D_2(y)$, 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:

Model	NN	SBR	DLM-NN	DLM
Accuracy	0.62	0.64	0.65	0.76

Table 2. Comparison of the accuracy metric on the PAIRS dataset using different models.

- 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 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 4.1 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.

4.2   Link Prediction in Knowledge Graphs

Neural-symbolic approaches have been proved to be very powerful to perform approximated logical reasoning [29]. A common approach is to assign to each logical constant and relation a learned vectorial representation [4]. 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. [28] 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 [5] 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. $\texttt{locatedIn}$(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 $\texttt{locatedIn}$ 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 $\texttt{locatedIn}$ and $\texttt{neighborOf}$, 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 [4], 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 $\boldsymbol{y}_t$ 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, once the model learns a rule stating the transitivity of the $\texttt{locatedIn}$ relation (i.e. $\texttt{locatedIn}(x, y) \land \texttt{locatedIn}(y, z) \rightarrow \texttt{locatedIn}(x, z)$). Exploiting instead the rule $\texttt{neighborOf}(x, y) \land \texttt{locatedIn}(y, z) \rightarrow \texttt{locatedIn}(x, z)$, the model should be capable of approximately solving task S2.

All 8 rules $\forall x \, \forall y \, \forall z \; \texttt{A}(x, y) \land \texttt{B(y,z)} \rightarrow \texttt{C}(y, z)$, where A, B and C are either $\texttt{neighborOf}$ or $\texttt{locatedIn}$ 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

Task	ComplEx	NTP	NTPλ	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 3. Comparison of the accuracy provided by different methods on link prediction on the Countries dataset. Bold numbers are the best performers for each task.

each task. The training process will iteratively minimize Equation 3 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 4.2 compares DLM against the state-of-the-art methods used by Rocktaschel et al. [28], namely ComplEx, NTP and NTPλ. 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.

5   Conclusions and future work

This paper presents Deep Logic Models that integrate (deep) learning and logic reasoning into a single fully differentiable architecture. The logic can be expressed with unrestricted FOL formalism, where each FOL rule is converted into a differentiable potential function, which can be integrated into the learning process. The main advantage of the presented framework is the ability to fully integrate learning from low-level representations and semantic high-level reasoning over the network outputs. Allowing to jointly learn the weights of the deep learners and the parameters controlling the reasoning enables a positive feedback loop, which is shown to improve the accuracy of both layers. Future work will try to bridge the gap between fully grounded methodologies like current Deep Logic Models and Theorem Provers which expand only the groundings needed to expand the frontier of the search space.

References

1. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. arXiv preprint arXiv:1505.04406 (2015)

2. Battaglia, P.W., Hamrick, J.B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al.: Relational inductive biases, deep learning, and graph networks. arXiv preprint arXiv:1806.01261 (2018)

3. Bengio, Y., et al.: Learning deep architectures for ai. Foundations and trends® in Machine Learning **2**(1), 1–127 (2009)

4. Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Advances in neural information processing systems. pp. 2787–2795 (2013)

5. Bouchard, G., Singh, S., Trouillon, T.: On approximate reasoning capabilities of low-rank vector spaces. AAAI Spring Syposium on Knowledge Representation and Reasoning (KRR): Integrating Symbolic and Neural Approaches (2015)

6. Chen, L.C., Schwing, A., Yuille, A., Urtasun, R.: Learning deep structured models. In: International Conference on Machine Learning. pp. 1785–1794 (2015)

7. De Raedt, L., Kimmig, A., Toivonen, H.: Problog: A probabilistic prolog and its application in link discovery. In: Proceedings of the 20th International Joint Conference on Artifical Intelligence. pp. 2468–2473. IJCAI'07, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2007), `http://dl.acm.org/citation.cfm?id=1625275.1625673`

8. Diligenti, M., Gori, M., Sacca, C.: Semantic-based regularization for learning and inference. Artificial Intelligence **244**, 143–165 (2017)

9. Donadello, I., Serafini, L., Garcez, A.d.: Logic tensor networks for semantic image interpretation. arXiv preprint arXiv:1705.08968 (2017)

10. Garcez, A.S.d., Broda, K.B., Gabbay, D.M.: Neural-symbolic learning systems: foundations and applications. Springer Science & Business Media (2012)

11. Giannini, F., Diligenti, M., Gori, M., Maggini, M.: On a convex logic fragment for learning and reasoning. IEEE Transactions on Fuzzy Systems (2018)

12. Goodfellow, I., Bengio, Y., Courville, A., Bengio, Y.: Deep learning, vol. 1. MIT press Cambridge (2016)

13. Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edn. (1994)

14. Hazan, T., Schwing, A.G., Urtasun, R.: Blending learning and inference in conditional random fields. The Journal of Machine Learning Research **17**(1), 8305–8329 (2016)

15. Hu, Z., Ma, X., Liu, Z., Hovy, E., Xing, E.: Harnessing deep neural networks with logic rules. arXiv preprint arXiv:1603.06318 (2016)

16. Kimmig, A., Bach, S., Broecheler, M., Huang, B., Getoor, L.: A short introduction to probabilistic soft logic. In: Proceedings of the NIPS Workshop on Probabilistic Programming: Foundations and Applications. pp. 1–4 (2012)

17. Lavrac, N., Dzeroski, S.: Inductive logic programming. In: WLP. pp. 146–160. Springer (1994)

18. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

19. Lin, G., Shen, C., Van Den Hengel, A., Reid, I.: Efficient piecewise training of deep structured models for semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3194–3203 (2016)

20. Ma, X., Hovy, E.: End-to-end sequence labeling via bi-directional lstm-cnns-crf. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). pp. 1064–1074. Association for Computational Linguistics (2016). https://doi.org/10.18653/v1/P16-1101, `http://aclweb.org/anthology/P16-1101`

21. Mahendran, A., Vedaldi, A.: Understanding deep image representations by inverting them. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 5188–5196 (2015)
22. Manhaeve, R., Dumančić, S., Kimmig, A., Demeester, T., De Raedt, L.: Deepproblog: Neural probabilistic logic programming. arXiv preprint arXiv:1805.10872 (2018)
23. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming **19**, 629–679 (1994)
24. Novák, V., Perfilieva, I., Mockor, J.: Mathematical principles of fuzzy logic, vol. 517. Springer Science & Business Media (2012)
25. Quinlan, J.R.: Learning logical definitions from relations. Machine learning **5**(3), 239–266 (1990)
26. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**(1-2), 107–136 (2006)
27. Rocktäschel, T., Riedel, S.: Learning knowledge base inference with neural theorem provers. In: Proceedings of the 5th Workshop on Automated Knowledge Base Construction. pp. 45–50 (2016)
28. Rocktäschel, T., Riedel, S.: End-to-end differentiable proving. In: Advances in Neural Information Processing Systems. pp. 3788–3800 (2017)
29. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: International Conference on Machine Learning. pp. 2071–2080 (2016)