LYRICS: a General Interface Layer to Integrate Logic Inference and Deep Learning[*]

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. In spite of the amazing results obtained by deep learning in many applications, a real intelligent behavior of an agent acting in a complex environment is likely to require some kind of higher-level symbolic inference. Therefore, there is a clear need for the definition of a general and tight integration between low-level tasks, processing sensorial data that can be effectively elaborated using deep learning techniques, and the logic reasoning that allows humans to take decisions in complex environments. This paper presents LYRICS, a generic interface layer for AI, which is implemented in TersorFlow (TF). LYRICS provides an input language that allows to define arbitrary First Order Logic (FOL) background knowledge. The predicates and functions of the FOL knowledge can be bound to any TF computational graph, and the formulas are converted into a set of real-valued constraints, which participate to the overall optimization problem. This allows to learn the weights of the learners, under the constraints imposed by the prior knowledge. The framework is extremely general as it imposes no restrictions in terms of which models or knowledge can be integrated. In this paper, we show the generality of the approach showing some use cases of the presented language, including model checking, supervised learning and collective classification.

Keywords: deep learning, prior knowledge injection, first order logic

1 Introduction

The success of deep learning relies on the availability of a large amount of supervised training data. This prevents a wider application of machine learning in real world applications, where the collection of training data is often a slow and expensive process, requiring an extensive human intervention. The introduction of prior-knowledge into the learning process is a fundamental step in overcoming these limitations. First, it does not require the training process to induce the

rules from the training set, therefore reducing the number of required training data. Secondly, the use of prior knowledge can be used to express the desired behavior of the learner on any input, providing better behavior guarantees in an adversarial or uncontrolled environment.

This paper presents LYRICS, a TensorFlow [1] environment based on a declarative language for integrating prior knowledge into machine learning, which allows the full expressiveness of First Order Logic (FOL) to define the knowledge. LYRICS has its root in frameworks like Semantic Based Regularization (SBR) [6,7] built on top of Kernel Machines and Logic Tensor Networks (LTN) [23] that can be applied to neural networks. These frameworks transform the FOL clauses into a set of constraints that are jointly optimized during learning. However, LYRICS generalizes both approaches by allowing to enforce the prior knowledge transparently at training and test time and dropping the previous limitations regarding the form of the prior knowledge. SBR and LTN are also hard to extend beyond classical classification tasks, where they have been applied in previous works, because the lack of a declarative front-end. On the other hand, LYRICS define a declarative language, dropping the barrier to build models exploiting the available domain knowledge in any machine learning context.

In particular, any many-sorted first-order logical theory can be expressed in the framework, allowing to declare domains of different sort, with constants, predicates and functions. LYRICS provides a very tight integration of learning and logic as any computational graph can be bound to a FOL predicate. This allows to constrain the learner both during training and inference. Since the framework is agnostic to the learners that are bound to the predicates, it can be used in a vast range of applications including classification, generative or adversarial ML, sequence to sequence learning, collective classification, etc.

1.1 Previous work

In the past few years many authors tackled specific applications by integrating logic and learning. Minervini et al. [16] proposes to use prior knowledge to correct the inconsistencies of an adversarial learner. Their methodology is designed ah-hoc for the tackled task, and limited to Horn clauses. A method to distill the knowledge in the weights of a learner is presented by Hu et al. [11], which is also based on a fuzzy generalization of FOL. However, the definition of the framework is limited to universally quantified formulas and to a small set of logic operators. Another line of research [21,5] attempts at using logical background knowledge to improve the embeddings for Relation Extraction. However, these works are also based on ad-hoc solutions that lack a common declarative mechanism that can be easily reused. They are all limited to a subset of FOL and they allow to injecting the knowledge at training time, with no guarantees that the output on the test set respect the knowledge.

Markov Logic Networks (MLN) [20] and Probabilistic Soft Logic (PSL) [13,2] 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 can

be trained independently, then frozen and stacked with the AI layer providing a higher-level inference mechanism. The language proposed in this paper instead allows to directly improve the underlying learner, while also providing the higher-level integration with logic. TensorLog [3] is a more recent framework to integrate probabilistic logical reasoning with the deep-learning infrastructure of TF, however TensorLog is limited to reasoning and does not allow to optimize the learners while performing inference. TensorFlow Distributions [9] and Edward [25] are also a related frameworks for integrating probability theory and deep learning. However, these frameworks focus on probability theory and not the representation of logic and reasoning.

2   The Declarative Language

LYRICS defines a TensorFlow (TF)[3] environment in which learning and reasoning are integrated. LYRICS provides a short number of basic constructs, which can be used to define the problem under investigation.

A *domain* determines a collection of individuals that share the same representation space and are analyzed and manipulated in a homogeneous way. For example, a domain can collect a set of 30×30 pixel images or the sentences of a book as bag-of-words.

```
Domain(label="Images")
```

Individuals (i.e. elements) can be added to their domain as follows:

```
Individual(label="Tweety", domain=("Images"), value=img0)
```

where *Tweety* is a label to uniquely identify a specific individual of the *Images* domain, represented by the image *img0*. This allows the user to directly reason about single individuals. The user can also provide a large amount of individuals without a specific label for each of them by specifying the tensor of their features during the domain definition.

A *function* can be defined to map elements from the input domains into an element of an output domain. A unary function takes as input an element from a domain and transforms it into an element of 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. The following example defines a function that returns a rotated image:

```
Function(label="rotate", domains=("Images"), function=RotateFunction)
```

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

A *predicate* can be defined as a function, mapping elements of the input domains to truth values. For example, a predicate *bird* determining whether an input patttern from the *Images* domain contains a bird and approximated by a neural network NN is defined as:

---

[3] https://www.tensorflow.org/

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

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 predicates. For instance, if we want to learn the previously defined predicate *bird* to be invariant to rotations, the user can express this knowledge by means of the following constraint:

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

Finally, any available *supervision* for the functions or predicates can be directly integrated into the learning problem. LYRICS provides a specific construct where this fitting is expressed, called *PointwiseConstraint*. This construct links to a computational graph where a loss is applied for each supervision. The loss defaults to the cross-entropy loss but it can be overridden to achieve a different desired behavior:

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

where `model` is a TF function like the *NN* function used before fitting the supervisions `labels` on the provided `inputs`.

3   From Logic to Learning

LYRICS transparently transforms a declarative description of the available knowledge applied to a set of objects into an optimization task. In this section, we show how the optimization algorithm is derived from its declaration.

Domains and Individuals. 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[4]. 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)`).

Functions and Predicates. FOL functions allow the mapping between individuals of the input domains to an individual of the output domain, i.e. $f_i : D^f_{i_1} \times \cdots \times D^f_{i_m} \to D^f_i$, where $D^f_{i_1}, \ldots, D^f_{i_m}$ are the input domains and

---

[4] 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.

t-norm op	Product	Lukasiewicz	Gödel
$x \wedge y$	$x \cdot y$	$\max(0, x + y - 1)$	$\min(x, y)$
$x \vee y$	$x + y - x \cdot y$	$\min(1, x + y)$	$\max(x, y)$
$\neg x$	$1 - x$	$1 - x$	$1 - x$
$x \Rightarrow y$	$x \leq y ? 1 : \frac{y}{x}$	$\min(1, 1 - x + y)$	$x \leq y ? 1 : y$

Table 1: Operations performed by the units of an expression tree given the inputs x, y and the used t-norm in the fundamental fuzzy logics.

D_i^f 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. $p_i : D_{i_1}^p \times \ldots \times D_{i_m}^p \rightarrow \{true, false\}$, where $D_{i_j}^p$ is the j-th domain of the i-th predicate. Functions and predicates are implemented using a TF architecture as explained in the previous section. 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 [7,23], 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.

Connectives and Quantifiers. Connectives and quantifiers are converted using the fuzzy generalization of FOL that was first proposed by Novak [19]. In particular, a *t-norm fuzzy logic* [10] generalizes Boolean logic to variables assuming values in $[0, 1]$. A t-norm fuzzy logic is defined by its t-norm that models the logical AND, and from which the other operations can be derived. Table 1 shows some possible implementation of the connectives using the fundamental t-norm fuzzy logics i.e. *Product, Łukasiewicz* and *Gödel* logics.

In general, formulas involve more than a predicate and are evaluated on the overall grounding vectors of such predicates. The way different evaluations of a certain formula are aggregated depend on the *quantifiers* occurring on its variables and their implementation. In particular, we consider the universal and

existential quantifier that can be seen as a logic AND and OR applied over each grounding of the data, respectively.

For instance, given a certain logical expression E with a universally quantified variable x_i can be calculated as the average of the t-norm generalization $t_E(\cdot)$, when grounding x_i over its groundings X_i:

$$\forall x_i \; E\big(\mathcal{P}(\mathcal{X})\big) \longrightarrow \Phi_\forall\big(\mathcal{P}(\mathcal{X})\big) = \frac{1}{|X_i|} \sum_{x_i \in X_i} t_E\big(\mathcal{P}(\mathcal{X})\big) \tag{1}$$

The truth degree of the existential quantifier is instead defined as the maximum of the t-norm expression over the domain of the quantified variable:

$$\exists x_i \; E\big(\mathcal{P}(\mathcal{X})\big) \quad \longrightarrow \quad \Phi_\exists\big(\mathcal{P}(\mathcal{X})\big) = \max_{x_i \in X_i} \; t_E\big(\mathcal{P}(\mathcal{X})\big) \tag{2}$$

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

Constraints. 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* [8]. 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 groundings (i.e. $f(\mathcal{X})$ or $p(\mathcal{X})$)
- 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 1 shows the translation of a logic formula into its expression tree and successively into a TensorFlow computational graph.

It is useful for the following to consider the real-valued constraint obtained by the described compilation process of the j-th logical rule and implemented by a TF graph as a parametric real function $\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)$. The function ψ_j takes as input the Cartesian product $\mathcal{X}_j$ of the domains of its quantified variables, returns the truth degree of the formula and it is parameterized by w_j^i, w_j^f and w_j^p, which are the sets of variable tensors related to the features of learnable individuals, to the parameters of learnable functions and to the parameters of learnable predicates, respectively. Let be $W^i = \{w_1^i, w_2^i, \dots\}$, $W^f = \{w_1^f, w_2^f, \dots\}$ and $W^p = \{w_1^p, w_2^p, \dots\}$.
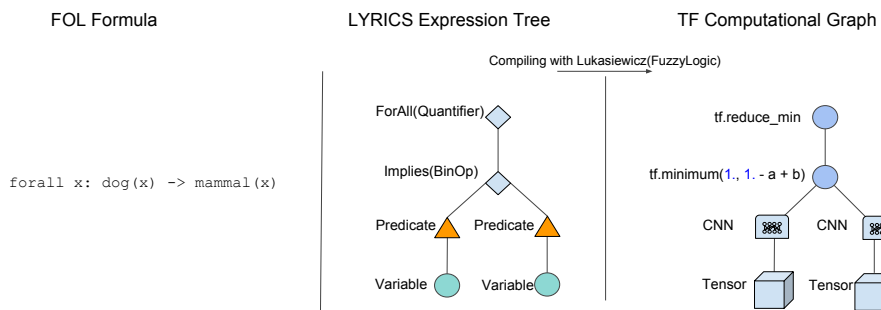
Fig. 1: The translation of the FOL formula $\forall x \; dog(x) \rightarrow mammal(x)$ into a Lyrics expression tree and then its mapping to a TF computational graph.

Optimization Problem. The goal of LYRICS is to build a learning process for some elements of interest (individuals, functions or predicates) by a declarative description of the desired behaviour of these elements. The desired behaviour is expressed by means of logical formulas. Thus, the optimization process is framed as finding the unknown elements which maximize the satisfaction of the set of logical formulas. Let $\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)$ indicate the real-valued constraint related to the j-th formula, as previously defined. Then, the derived optimization problem is:

$$\max_{W^i, W^f, W^p} \sum_{j=1}^{T} \lambda_c^j \psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p),$$

where λ_c^j denotes the weight for the j-th logical constraint. These weights are considered hyper-parameters of the model and are provided by the user during constraint definition. The maximization problem can be translated into a minimization problem as follows:

$$\min_{W^i, W^f, W^p} \sum_{j=1}^{T} \lambda_c^j \mathcal{L}\Big(\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)\Big),$$

Here, the function $\mathcal{L}$ represents any monotonically decreasing transformation of the constraints conveniently chosen according to the problem under investigation. In particular, we may exploit the following mappings:

$$
\begin{aligned}
\textbf{(a)} \;\; & \mathcal{L}\Big(\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)\Big) = 1 - \psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p) \;, \\
\textbf{(b)} \;\; & \mathcal{L}\Big(\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)\Big) = - \log\Big(\psi_j(\mathcal{X}_j; w_j^i, w_j^f, w_j^p)\Big) \;.
\end{aligned}
\tag{3}
$$

These specific choices for the function $\mathcal{L}$ are directly related to the Łukasiewicz and Product t-norms, indeed they are *additive generators* for these t-norm fuzzy logics. For more details on generated t-norms we recommend e.g. [14].

4   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. The software of the framework and the experiments are made available at "provided as supplemental materials".

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 2(a). The remaining points are split into 200 unsupervised training points, shown in Figure 2(b) 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)
```

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 predicate determining if two patterns are close according to a validated threshold of the Euclidean distance. The training is then re-executed starting from the same initial conditions as in the supervised-only case.

Figure 2(c) shows the class assignments of the patterns in the test set, when using only classical learning from supervised examples. Finally, Figure 2(d) presents the assignments when learning from examples and constraints.
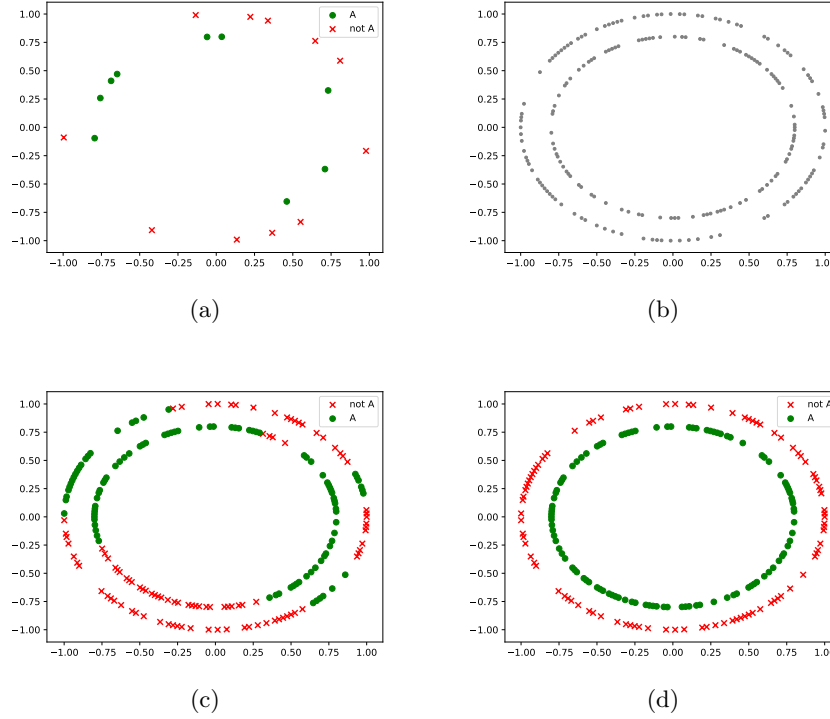
Fig. 2: Semi-supervised Learning: (a) data that is provided with a positive and negative supervision for class A; (b) the unsupervised data provided to the learner; (c) class assignments using only the supervised examples; (d) class assignments using learning from examples and constraints.

Collective Classification. Collective classification [22] 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 3(a). The examples are partially labeled and drawn from a uniform distribution on both the positive and negative regions for all the classes.

In a first stage, the classifiers for the three classes are trained in a supervised fashion using a two-layer neural network taking four positive and four negative examples for each class. This is implemented via the following declaration:

```
Domain(label="Points", data=X)
```
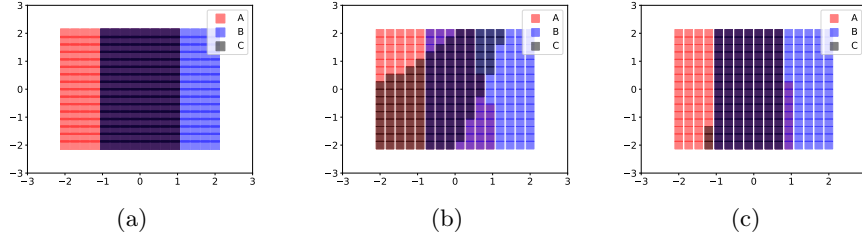
Fig. 3: Collective classification: (a) classes assignments; (b) the predictions after the supervised step; (c) the predictions with collective classification and rules satisfaction (best viewed in colors).

```
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 3(b). 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 as close as possible to the initial classifier predictions, acting as priors, but also respecting 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 classifier outputs.
PointwiseConstraint(A, priorsA, X_test)
PointwiseConstraint(B, priorsB, X_test)
PointwiseConstraint(C, priorsC, X_test)
```

where X_test is the set of test datapoints and priorsA, priorsB, priorsC denote the predictions of the classifiers to which the final assignments have to stay close. As we can see from Fig.3(c), the collective step fixes some wrong predictions.

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: $\mathbf{A} = [-2, 2] \times [-2, 2]$, $\quad \mathbf{B} = [-1, 1] \times [-1, 1]$. 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 [17].

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. [4], many papers have applied deep neural networks to text chunking. In this paper, 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 [24] 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. [12] 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, $\forall\, x \,\forall\, t\; BNP(x,t) \Rightarrow \neg IVP(x,t+1) \land \neg IPP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$ 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 the output consistency can be expressed in FOL using the following statements:

$$\forall\, x \,\forall\, t\; BNP(x,t) \Rightarrow \neg IVP(x,t+1) \land \neg IPP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; BVP(x,t) \Rightarrow \neg INP(x,t+1) \land \neg IPP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; BPP(x,t) \Rightarrow \neg IVP(x,t+1) \land \neg INP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; INP(x,t) \Rightarrow [\neg IVP(x,t+1) \land \neg IPP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; IVP(x,t) \Rightarrow [\neg INP(x,t+1) \land \neg IPP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; IPP(x,t) \Rightarrow [\neg IVP(x,t+1) \land \neg INP(x,t+1) \land \neg IADVP(x,t+1) \land \dots$$
$$\forall\, x \,\forall\, t\; INP(x,t+1) \Rightarrow BNP(x,t) \lor INP(x,t)$$
$$\forall\, x \,\forall\, t\; IVP(x,t+1) \Rightarrow BVP(x,t) \lor IVP(x,t)$$
$$\forall\, x \,\forall\, t\; IPP(x,t+1) \Rightarrow BPP(x,t) \lor IPP(x,t)$$
$$\dots$$

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 2 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.

Document Classification on the Citeseer dataset. This section applies the proposed framework to a standard ML dataset. The CiteSeer dataset[5] [15] 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

---

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

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

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 3 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 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 4 compares the neural network classifiers against other two content-based classifiers, namely logistic regression

	% 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 3: 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 4: 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.

(LR) and Naive Bayes (NB), and against collective classification approaches using network data: Iterative Classification Algorithm (ICA) [18] and Gibbs Sampling (GS) [15] 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) [22] and Relaxation Labeling through Mean-Field Approach (MF) [22] 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   Conclusions

This paper presents a novel and general framework, called LYRICS, to bridge logic reasoning and deep learning. The framework is directly implemented in

TensorFlow, allowing a seaming-less integration that is architecture agnostic. The frontend of the framework is a declarative language based on First–Order Logic. Throughout the paper are presented a set of examples illustrating the generality and expressivity of the framework, which can be applied to a large range of tasks.

Future developments of the proposed framework include a learning mechanism of the weights of the constraints. This would allow to consider more general rule schemata that will be weighted with coefficients automatically learned by the parameter optimization according to the degree of satisfaction of any rule. This will improve the framework especially to deal with soft constraints expressing some statistical co-occurrence among the classes involved in the learning problem. Moreover, the differentiability of fuzzy logic could suggest new methods for learning a set of constraints in logical form that may be understandable.

References

1. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al.: Tensorflow: A system for large-scale machine learning. In: OSDI. vol. 16, pp. 265–283 (2016)
2. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. arXiv preprint arXiv:1505.04406 (2015)
3. Cohen, W.W.: Tensorlog: A differentiable deductive database. arXiv preprint arXiv:1605.06523 (2016)
4. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., Kuksa, P.: Natural language processing (almost) from scratch. Journal of Machine Learning Research **12**(Aug), 2493–2537 (2011)
5. Demeester, T., Rocktäschel, T., Riedel, S.: Lifted rule injection for relation embeddings. arXiv preprint arXiv:1606.08359 (2016)
6. Diligenti, M., Gori, M., Maggini, M., Rigutini, L.: Bridging logic and kernel machines. Machine learning **86**(1), 57–88 (2012)
7. Diligenti, M., Gori, M., Saccà, C.: Semantic-based regularization for learning and inference. Artificial Intelligence (2015)
8. Diligenti, M., Roychowdhury, S., Gori, M.: Image classification using deep learning and prior knowledge. In: Proceedings of Third International Workshop on Declarative Learning Based Programming (DeLBP) (February 2018)
9. Dillon, J.V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., Saurous, R.A.: Tensorflow distributions. arXiv preprint arXiv:1711.10604 (2017)
10. Hájek, P.: Metamathematics of fuzzy logic, vol. 4. Springer Science & Business Media (1998)
11. Hu, Z., Ma, X., Liu, Z., Hovy, E., Xing, E.: Harnessing deep neural networks with logic rules. arXiv preprint arXiv:1603.06318 (2016)
12. Huang, Z., Xu, W., Yu, K.: Bidirectional lstm-crf models for sequence tagging. arXiv preprint arXiv:1508.01991 (2015)
13. 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)
14. Klement, E.P., Mesiar, R., Pap, E.: Triangular norms. position paper ii: general constructions and parameterized families. Fuzzy Sets and Systems **145**(3), 411–438 (2004)

15. Lu, Q., Getoor, L.: Link-based classification. In: Proceedings of the 20th International Conference on Machine Learning (ICML-03). pp. 496–503 (2003)
16. Minervini, P., Demeester, T., Rocktäschel, T., Riedel, S.: Adversarial sets for regularising neural link predictors. arXiv preprint arXiv:1707.07596 (2017)
17. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming **19**, 629–679 (1994)
18. Neville, J., Jensen, D.: Iterative classification in relational data. In: Proc. AAAI-2000 Workshop on Learning Statistical Models from Relational Data. pp. 13–20 (2000)
19. Novák, V., Perfilieva, I., Močkoř, J.: Mathematical principles of fuzzy logic (1999)
20. Richardson, M., Domingos, P.: Markov logic networks. Machine learning **62**(1), 107–136 (2006)
21. Rocktäschel, T., Singh, S., Riedel, S.: Injecting logical background knowledge into embeddings for relation extraction. In: Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 1119–1129 (2015)
22. Sen, P., Namata, G., Bilgic, M., Getoor, L., Galligher, B., Eliassi-Rad, T.: Collective classification in network data. AI magazine **29**(3), 93 (2008)
23. Serafini, L., Garcez, A.S.d.: Learning and reasoning with logic tensor networks. In: AI* IA. pp. 334–348 (2016)
24. Tjong Kim Sang, E.F., Buchholz, S.: 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. pp. 127–132. Association for Computational Linguistics (2000)
25. Tran, D., Hoffman, M.D., Saurous, R.A., Brevdo, E., Murphy, K., Blei, D.M.: Deep probabilistic programming. In: International Conference on Learning Representations (2017)