Learning First-Order Rules with Differentiable Logic Program Semantics

Kun Gao[1] , **Katsumi Inoue**[2] , **Yongzhi Cao**[1] and **Hanpin Wang**[3,1*]
[1]Key Laboratory of High Confidence Software Technologies (MOE), School of Computer Science, Peking University
[2]National Institute of Informatics
[3]School of Computer Science and Cyber Engineering, Guangzhou University
kungao@pku.edu.cn, inoue@nii.ac.jp, {caoyz, whpxhy}@pku.edu.cn

Abstract

Learning first-order logic programs (LPs) from relational facts which yields intuitive insights into the data is a challenging topic in neuro-symbolic research. We introduce a novel differentiable inductive logic programming (ILP) model, called differentiable first-order rule learner (DFOL), which finds the correct LPs from relational facts by searching for the interpretable matrix representations of LPs. These interpretable matrices are deemed as trainable tensors in neural networks (NNs). The NNs are devised according to the differentiable semantics of LPs. Specifically, we first adopt a novel propositionalization method that transfers facts to NN-readable vector pairs representing interpretation pairs. We replace the immediate consequence operator with NN constraint functions consisting of algebraic operations and a sigmoid-like activation function. We map the symbolic forward-chained format of LPs into NN constraint functions consisting of operations between subsymbolic vector representations of atoms. By applying gradient descent, the trained well parameters of NNs can be decoded into precise symbolic LPs in forward-chained logic format. We demonstrate that DFOL can perform on several standard ILP datasets, knowledge bases, and probabilistic relation facts and outperform several well-known differentiable ILP models. Experimental results indicate that DFOL is a precise, robust, scalable, and computationally cheap differentiable ILP model.

1 Introduction

Nowadays, knowledge discovery is an important technique for people acquiring knowledge from either large or complex realistic datasets. Relational data mining constructs a human-readable representation from relational datasets. An explicit logic program (LP) can be a clear explanation to complex, incomplete, or noisy relational datasets due to the limitations of current techniques, especially in the datasets from business, biology, and medicine. However, learning high accuracy LPs from realistic relational datasets is still a hardcore problem in the field of machine learning.

Inductive logic programming (ILP) is firstly proposed by [Muggleton, 1991] as a combination of inductive learning and logic programming technique has contained several methods. The purely symbolic ILP methods [Muggleton *et al.*, 2012] learning LPs typically support lifelong learning and have more explainability [Cropper *et al.*, 2020]. However, some purely symbolic approaches fail in ambiguous realistic facts. By employing neural networks (NNs), differentiable ILP models generate explicit logic rules in a fast and precise manner from realistic relational facts. However, NN-based ILP models usually require many examples [Dong *et al.*, 2019] to learn concepts, while symbolic ILP methods only need a few examples [Cropper *et al.*, 2020].

In this paper, we propose a differentiable ILP method called differentiable first-order rule learner (DFOL), which generates first-order LPs from positive examples and background assumptions without logic templates. In DFOL, a novel propositionalization method transfers relational facts into NN-readable data, i.e., vector pairs representing interpretation pairs. Besides, the matrix embeddings of LPs are regarded as the trainable parameters in NNs. Based on the differentiable immediate consequence operator and the forward-chained format of LPs, we design some differentiable constraint functions to guide the NNs to find the correct embeddings of LPs. Finally, DFOL demonstrates a high degree of interpretability as the parameters in NNs can be decoded to human-readable symbolic rules. Simultaneously, the correct symbolic rules can be encoded to the fixed parameters in NNs to facilitate the training process.

The main contributions are summarized below: (1) A flexible, precise, fast, and robust differentiable first-order rule learning model is proposed. (2) The proposed model is interpretable. We can not only extract symbolic LPs from the model but also embed LPs into NNs. (3) We demonstrate that our method outperforms the baselines on most datasets, including small ILP datasets and large knowledge bases.

Related Work. Neuro-symbolic models have been focused on over the past decades. For learning propositional LPs, d'Avila Garcez *et al.* [2001] and Lehmann *et al.* [2010] designed algorithms to extract propositional LPs from NNs. On the other hand, Gao *et al.* [2021] generated proposi-

*Contact Author
Full version available at https://arxiv.org/abs/2204.13570

tional LPs using NNs from input-output pairs of the immediate consequence operator of LPs by making the symbolic method proposed by Inoue *et al.* [2014] be differentiable. However, compared with first-order LPs, propositional LPs have less ability to describe relational facts. For first-order LPs, the similar work includes [Evans *et al.*, 2021; Evans and Grefenstette, 2018; Rocktäschel and Riedel, 2017; Sourek *et al.*, 2018]. In their models, the explicit LPs are learned based on the given templates. These models need to learn the weights given rules or fill the predicates in rule templates. However, we construct LPs without any explicit templates but follow the forward-chained format. Thus, DFOL is able to find the correct rules in a flexible way, which means that we do not need any prior knowledge about the tasks, and the generated LPs may have more diverse forms than those generated from strongly biased templates. From the perspective of the representations of LPs, Qu *et al.* [2021] regarded LPs as latent variables. They developed an EM-based algorithm for extracting LPs through a rule generator and a reasoning predictor. Kaur *et al.* [2019] regarded a rule as a lifted random walk. In contrast to them, DFOL uses small matrix embeddings to encode LPs. Hence, with the help of the differentiable semantics of LPs, NNs can be adopted in DFOL to search the embeddings fastly and robustly. When adopting NNs to get LPs, Yang *et al.* [2017] used differentiable operations to learn the embeddings of LPs. However, the embeddings of LPs in DFOL are more interpretable, and we do not need any algorithms to transfer subsymbolic matrices into symbolic LPs. CILP++ proposed by França *et al.* [2014] uses the bottom clauses propositionalization method and three-layer NNs to imitate the immediate consequence operator of LPs. However, CILP++ cannot be interpreted into symbolic LPs directly. Similarly, Teru *et al.* [2020] and Hohenecker and Lukasiewicz [2020] used embedding methods to perform relation prediction tasks, which are also induction tasks but do not generate explicit LPs as the results. Excepting these induction tasks, d'Avila Garcez and Zaverucha [1999] and Serafini and d'Avila Garcez [2016] considered deduction tasks with NNs and pre-defined LPs.

2 Preliminaries

We recall the concepts of LPs, ILP, propositionalization methods, and differentiable semantics of LPs in this section.

2.1 Logic Programs

A (definite) LP P consists of several rules, and each rule r is described as: $\alpha_h \leftarrow \alpha_1 \wedge \alpha_2 \cdots \wedge \alpha_n$ $(n \geq 0)$, where α_h is the head atom denoted as $head(r)$; α_i's $(0 \leq i \leq n)$ are the body atoms, and the conjunction $\alpha_1 \wedge \alpha_2 \cdots \wedge \alpha_n$ is the body of r. The set of all body atoms of a rule r is denoted as $body(r)$. A set of rules with the same head atom α_h called a same head logic program (SHLP): $\alpha_h \leftarrow \beta_1$, $\alpha_h \leftarrow \beta_2$, ..., $\alpha_h \leftarrow \beta_m$, can be identified with a rule of the form: $\alpha_h \leftarrow \beta_1 \vee \beta_2 \vee \ldots \vee \beta_m$, where each β_i is the body of the i-th rule and $\beta_1 \vee \beta_2 \vee \ldots \vee \beta_m$ is a disjunction of conjunctions of literals, i.e., a disjunction normal form formula. In first-order LPs, each atom α is a tuple $p(t_1, t_2, \ldots, t_n)$, where p indicates n-ary predicate and $t_1, t_2 \ldots, t_n$ are terms, and a term is either a variable or a constant. When an atom has no variable, the atom is a ground atom. A ground rule is derived based on substitutions, where all variables are replaced by constants. We use uppercase letters for variables and lowercase letters for constants. Let B be a Herbrand base, including all ground atoms, and an interpretation I is a subset of B. For an LP P and an interpretation I, the immediate consequence operator $T_P : 2^B \rightarrow 2^B$ [Van Emden and Kowalski, 1976] describes the rules with interpretations: $T_P(I) = \{head(r) \mid r \in g(P), body(r) \subseteq I\}$, where $g(P)$ is the ground LP based on P. Hence, given an interpretation I as the current state, $T_P(I)$ is regarded as an interpretation as the next state, which is the set of ground atoms that are derived from the rules of P under the condition that the atoms in I are true. A forward-chained format [Kaminski *et al.*, 2018] usually governs the format of an LP, which specifies that the body of rules should satisfy that the variables in the head atom are connected by a binary atomic chain:

$$p_t(X, Y) \leftarrow p_1(X, Z_1) \wedge p_2(Z_1, Z_2) \wedge \cdots \wedge p_{n+1}(Z_n, Y). \quad (1)$$

The variable depth indicates the number of variables not appearring in the head atom. For example, the variable depth is n in the rule (1).

2.2 Inductive Logic Programming and Propositionalization

An ILP task aims at generating an LP P headed by a target atom α_t given a tuple $(\mathcal{B}, \mathcal{P}, \mathcal{N})$. Let p_t denote a target predicate; $\mathcal{B}$ is a set of ground atoms called background assumptions; $\mathcal{P}$ is a set of positive ground atoms, taken from the ground of the target atom; $\mathcal{N}$ is a set of negative ground atoms, taken outside the ground of the target atom. Formally, a solution P of an ILP task is:

$$\mathcal{B}, P \models e^+, e^+ \in \mathcal{P}; \;\; \mathcal{B}, P \not\models e^-, e^- \in \mathcal{N}.$$

When learning first-order LPs, the propositionalization method [Kramer *et al.*, 2001] is an effective way to transform relational data into attribute-valued data. After applying propositionalization methods, we can use NNs to process the attribute-valued data and learn LPs from the relational facts. In [França *et al.*, 2014], the propositionalization method transfers relational facts to unground atoms with Boolean values. These unground atoms with values are called first-order features or features for short.

2.3 Differentiable Semantics of Logic Programs

In this section, we show the matrix representations and differentiable semantics of LPs. Let P be an SHLP with a head atom α_h, n different body atoms, and m different rules. Then P is represented by an SH matrix $\mathbf{M}_P \in [0, 1]^{m \times n}$. Each element a_{kj} in $\mathbf{M}_P$ is defined as follows [Gao *et al.*, 2021]:

1. $a_{kj_i} = l_i$, where $l_i \in (0, 1)$ and $\sum_{s=1}^{p} l_s = 1$ $(1 \leq i \leq p,\ 1 \leq j_i \leq n,\ 1 \leq k \leq m)$, if the k-th rule is $\alpha_h \leftarrow \alpha_{j_1} \wedge \cdots \wedge \alpha_{j_p}$;
2. $a_{kh} = 1$, if the k-th rule is $\alpha_h \leftarrow \alpha_h$;
3. $a_{kj} = 0$, otherwise.

In fact, each row in $\mathbf{M}_P$ corresponds to a rule in P, and each non-zero value in a row of $\mathbf{M}_P$ corresponds to a body atom

Algorithm 1 The propositionalization method in DFOL

Input: A variable set $V = \{X, Y, V_1, V_2, \ldots, V_d\}$ with the variable depth d; target atom α_t, e.g., binary predicate $p_t(X, Y)$ or unary predicate $p_t(X)$; body features set P_F; training positive examples $\mathcal{P}$ and training fact set F.
Output: A trainable dataset T.

(Preparation Process)
Let $\mathrm{X}, \mathrm{Y}, \mathrm{V}_1, \mathrm{V}_2, \ldots, \mathrm{V}_d$ represent the domains of variables $X, Y, V_1, V_2, \ldots, V_d$.
If the target predicate p_t is binary, then for each positive example $p_t(o_1, o_2) \in \mathcal{P}$, add o_1 and o_2 to the sets X and Y, respectively. Besides, add all entities in F to the sets $\mathrm{V}_1, \mathrm{V}_2, \ldots, \mathrm{V}_d$. If p_t is unary, for each positive example $p_t(o_1) \in \mathcal{P}$, add o_1 to the set X. Besides, add all entities in F to the sets Y, $\mathrm{V}_1, \mathrm{V}_2, \ldots, \mathrm{V}_d$.
(Generation Process)
Calculate the set S with all substitutions: $S = \mathrm{X} \times \mathrm{Y} \times \mathrm{V}_1 \times \mathrm{V}_2 \times \cdots \times \mathrm{V}_d$.
for each $\theta_k = \{x_k/X, y_k/Y, v_1^k/V_1, \ldots, v_d^k/V_d\} \in S$ **do**
    Initialize that $\mathbf{v}_i^k = \mathbf{0}$ and $\mathbf{v}_o^k = [0]$. Under the current substitution θ_k, for each $\alpha_j \in P_F$, $\mathbf{v}_i^k[j] = 1$ if $g(\alpha_j) \in F$; Then $\mathbf{v}_o^k = [1]$ if $g(\alpha_t) \in \mathcal{P}$. Next, add the pair of interpretation vectors $(\mathbf{v}_i^k, \mathbf{v}_o^k)$ to T.
end for
(Examination Process)
For each data in T, delete the data $(\mathbf{v}_i^k, \mathbf{v}_o^k)$ iff $\mathbf{v}_i^k = \mathbf{0}$.
Discard the m-th body feature and the corresponding values in all input interpretation vectors iff for all $k \in [1, |T|]$, $\mathbf{v}_i^k[m] = 0$ ($m \in [1, |P_F|]$) holds.
return T

in the corresponding rule in P. Example 1 of Appendix A shows the SH matrix corresponding to an SHLP. Let $\mathbf{M}[k, \cdot]$ and $\mathbf{M}'[k, \cdot, \cdot]$ denote the k-th row in the matrix $\mathbf{M}$ and k-th matrix of the three-dimensional tensor $\mathbf{M}'$, respectively. Moreover, an interpretation vector $\mathbf{v} = \{a_1, \ldots, a_n\}^T$ represents an interpretation in the vector space. If the Boolean value of α_k is *True*, then $a_k = 1$; otherwise, $a_k = 0$. We also use $\mathbf{v}[k]$ to denote the k-th element in the vector $\mathbf{v}$. Then, the immediate consequence operator can be represented in the vector space by Equation (2a) [Sakama *et al.*, 2021]. When $x \geq 1$, then the threshold function $\theta(x) = 1$; otherwise, $\theta(x) = 0$. To employ an NN to learn the SH matrix of an LP, a differentiable logic semantics used in [Gao *et al.*, 2021] replaces the logical or operator with the product t-norm and uses the differentiable function $\phi(x - 1)$ in Equation (2b) to replace the $\theta(x)$ function. The hyperparameter γ controls the slope similarity between the functions ϕ and θ.

$$\mathbf{v}_o = \vee_{k=1}^{m} \theta(\mathbf{M}_P[k, \cdot] \times \mathbf{v}_i^T), \quad (2a) \qquad \phi(x) = \frac{1}{1 + e^{-\gamma x}}. \quad (2b)$$

3 A Neural Network-Based Rule Learner

3.1 Propositionalization

In this section, we describe the propositionalization method in Algorithm 1. We follow the Closed-World Assumption

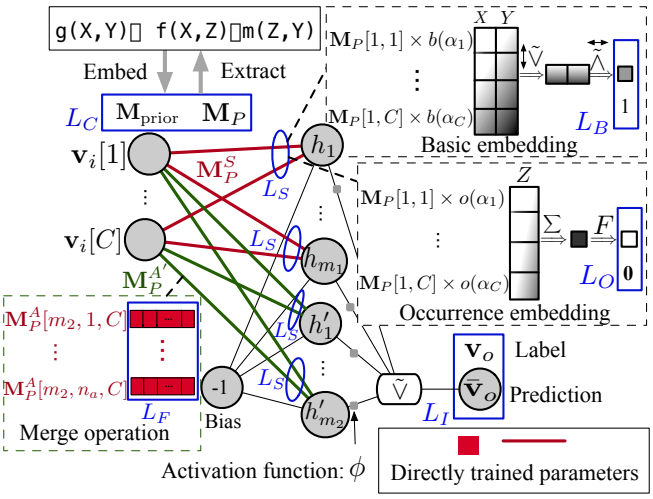

Figure 1: The architecture of DFOL. The logic program, basic and occurence embeddings are instantiated under the *grandparent* task.

that any example that does not appear in the set $\mathcal{P}$ is in the set $\mathcal{N}$. The training fact set F includes training positive examples and training background assumptions. Assume that each predicate in a task is either binary or unary. By scanning the dataset once, the number of binary predicates n_b and the number of unary predicates n_u can be determined. When the body of a rule includes the target atom, the rule is a tautology, which we discard when describing relational facts. For generating an SHLP P headed by a target atom, the set of possible body features P_F includes all possible binary and unary atoms except the target atom. Then, we have $|P_F| = \mathrm{A}(|V|, 2) \times n_b + |V| \times n_u - 1$, where $\mathrm{A}(|V|, 2)$ is the number of arrangements of 2 items from $|V|$ variables.

After the propositionalization, the relational facts are transformed into pairs of interpretation vectors $(\mathbf{v}_i, \mathbf{v}_o)$. The features in each $\mathbf{v}_i$ are considered as valid features, and let C represent $|\mathbf{v}_i|$. An input interpretation vector $\mathbf{v}_i$ corresponds to I_i, which includes all the values of the valid body features in P. An output interpretation vector $\mathbf{v}_o$ corresponds to I_o, which determines the value of the head feature in the SHLP P. Example 2 of Appendix A under the *predecessor* (*pre*) relation illustrates the proposed propositionalization method.

We analyze the correctness of the propositionalization method as follows: In fact, if all the ground body atoms in the LP P describing a relational dataset are satisfied under a substitution θ, the Boolean value of the ground head atom under θ must be *True*. After applying all possible substitutions on all first-order features for a relation dataset, we can generate pairs of interpretations (I_i, I_o) that correspond to interpretation vectors $(\mathbf{v}_i, \mathbf{v}_o)$ and satisfy the relation $I_o = T_P(I_i)$. Therefore, with the help of the robustness of NNs, we can learn the most important body features when the target feature is *True*. Moreover, the complexity of the propositionalization method is $O(|S| \times |P_F|)$, where S is defined in Algorithm 1.

3.2 The Neural Networks in DFOL

In this section, we describe the proposed NNs and constraint functions depicted in Figure 1. Taking the training data $(\mathbf{v}_i,$

$\mathbf{v}_o) \in T$ as the input, an NN learns the SH matrix $\mathbf{M}_P$ encoding an LP P. The LP P meets the forward-chained format described in the rule (1) and the immediate consequence operator $I_o = T_P(I_i)$, where I_i and I_o correspond to $\mathbf{v}_i$ and $\mathbf{v}_o$, respectively. First, we use a matrix $\mathbf{M}_P^S \in [0,1]^{m_1 \times C}$ as a trainable tensor to encode P, where m_1 is a hyperparameter describing the number of logic rules. Besides, we use $\mathbf{M}_P^A \in [0,1]^{m_2 \times n_a \times C}$ as another trainable tensor to encode P, where m_2 and n_a are hyperparameters. Then, we define the merge operation and concatenation operation as follows:

$$\mathbf{M}_P^{A'} = \frac{1}{n_a}\sum_{i=1}^{n_a} \mathbf{M}_P^A[\cdot, i, \cdot],\ \mathbf{M}_P = \text{concat}(\mathbf{M}_P^S, \mathbf{M}_P^{A'}),$$

where tensors in the arguments of concat function are joined along the vertical dimension. Besides, $\mathbf{M}_P^{A'} \in [0,1]^{m_2 \times C}$ and $\mathbf{M}_P \in [0,1]^{m \times C}$, where $m = m_1 + m_2$. We illustrate the merge and concat functions in Example 3 of Appendix A. We replace the immediate consequence operator of P with a differentiable inference and define an inference loss as follows:

$$\bar{\mathbf{v}}_o = \widetilde{\vee}_{k=1}^m(\phi(\mathbf{M}_P[k,\cdot] \times \mathbf{v}_i^T - 1)),\ L_I = H(\bar{\mathbf{v}}_o, \mathbf{v}_o),$$

where the activation function ϕ is defined in Equation (2b). The function H denotes the binary cross-entropy function. The symbol $\tilde{\vee}$ denotes the differentiable fuzzy or operation [Hájek, 1998]. Compared with other fuzzy logic semantics, we use product t-norm as the fuzzy logic semantics in DFOL for avoiding a zero gradient [Evans and Grefenstette, 2018]:

$$\widetilde{\vee}_{i=1}^n x_i = 1 - \prod_{i=1}^n (1 - x_i),\ \ \widetilde{\wedge}_{i=1}^n x_i = \prod_{i=1}^n x_i$$

Now, we analyze the roles of the tensor $\mathbf{M}_P^A$. Compared with $\mathbf{M}_P^S$, $\mathbf{M}_P^A$ has more parameters. Besides, the merge operation keeps the numbers of columns including the non-zero values from $\mathbf{M}_P^A[k,\cdot,\cdot]$ to $\mathbf{M}_P^{A'}[k,\cdot]$. Thus, when $\mathbf{M}_P^{A'}[k,\cdot]$ encodes a correct rule r_k headed by the target atom, the rows in the matrix $\mathbf{M}_P^A[k,\cdot,\cdot]$ have the opportunity to represent rules headed by auxiliary predicates [Muggleton *et al.*, 2015]. The body of a rule headed by an auxiliary predicate may include a part of the body atoms that appear in the rule r_k. Hence, the matrix $\mathbf{M}_P^A$ boosts the training process. Example 4 of Appendix A describes the process of auxiliary predicate invention. Hence, the parameters m_2 and n_a indicate the number of rules headed by the target atom and the number of rules headed by auxiliary predicates, respectively.

Then, we describe other essential constraint functions to generate precise LPs. Firstly, we use a sum loss function L_S according to the property described in Section 2.3 that the sum of each row in an SH matrix is equal to one:

$$L_S = \sum_{k=1}^m \text{MSE}(\sum_i^C \mathbf{M}_P[k,i], 1),$$

where MSE is the mean square error loss function. To make the rules extracted from $\mathbf{M}_P$ meet the forward-chained format defined in the rule (1), we stipulate two spatial constraints:

1. Basic constraint: The body of each rule in an LP contains all variables that appear in the head atom.
2. Occurrence constraint: In the body of each rule, the number of occurrences of each variable that does not appear in the head atom is not one.

Then, we give each body atom a basic embedding and an occurrence embedding, and devise basic loss function and occurrence loss function to implement the above spatial constraints. Suppose that the arity of the head atom in an LP P is t, then the variable depth is $|V| - t$. Let V_h and V_o be the variable sets with elements appearing in the head atom and not appearing in the head atom, respectively. Let $b(\alpha) \in \{0,1\}^t$ and $o(\alpha) \in \{0,1\}^{|V|-t}$ be the basic and occurrence embeddings corresponding to the body atom α, respectively. If the i-th variable in V_h (or V_o) appears in α, then $b(\alpha)[i]$=1 (or $o(\alpha)[i]$=1); otherwise, $b(\alpha)[i]$=0 (or $o(\alpha)[i]$=0). Example 5 of Appendix A illustrates a basic embedding and an occurrence embedding. Then, we devise the following functions to implement the basic constraint:

$$\mathbf{M}_b^k = \text{concat}(\mathbf{M}_P[k,1] \times b(\alpha_1), \mathbf{M}_P[k,2] \times b(\alpha_2), \ldots, \mathbf{M}_P[k,C] \times b(\alpha_C))$$

where α_i is the i-th valid first-order feature. The matrix $\mathbf{M}_b^k \in [0,1]^{C \times t}$ includes the occurrence information of all variables in V_h across all valid features for the k-th rule in an LP. Then, we use the fuzzy conjunction and disjunction operators to calculate the possibility that all variables in V_h appear in the k-th rule at once, and we define a basic loss function L_B:

$$\text{basic}_k = \widetilde{\wedge}_{j=1}^t \widetilde{\vee}_{i=1}^C \mathbf{M}_b^k[i,j],\ L_B = \sum_{k=1}^m \text{MSE}(\text{basic}_k, 1).$$

To implement the occurrence constraint, we use the following equation to concatenate all possibilistic occurrence embeddings across all valid features to $\mathbf{M}_o^k \in [0,1]^{C \times (|V|-t)}$:

$$\mathbf{M}_o^k = \text{concat}(\mathbf{M}_P[k,1] \times o(\alpha_1), \mathbf{M}_P[k,2] \times o(\alpha_2), \ldots, \mathbf{M}_P[k,C] \times o(\alpha_C)).$$

Then, the possibility for each variable in V_o across all the valid features in the k-th rule are summarized into the matrix $\mathbf{V}_o^k = \sum_{i=1}^C \mathbf{M}_o^k[i,\cdot]$. Hence, $\mathbf{V}_o^k \in [0,1]^{|V|-t}$ includes the possibility of each variable in V_o in the k-th rule. Then, we set a measurement function $F(x) : \mathbb{N} \to [0,a]$ to reflect the number of occurrences of each variable in V_o, and define an occurrence loss function L_O as follows:

$$F(x) = a \cdot e^{b - c(x-d)^2},\ L_O = \sum_{k=1}^m F(\mathbf{V}_o^k),$$

where a, b, c, and d are hyperparameters. As x gets closer to d, the value of $F(x)$ gets larger; otherwise, $F(x)$ gets closer to 0. Example 6 of Appendix A illustrates the operations related to a basic and occurrence embedding.

Next, we consider the cosine similarity, $\cos(\mathbf{u}, \mathbf{v}) = \frac{\mathbf{u} \cdot \mathbf{v}}{|\mathbf{u}||\mathbf{v}|}$, denoting the similarity between two vectors. When two vectors have greater dissimilarity, their cosine similarity is close to -1; otherwise, their cosine similarity is close to 1. Since the parameter n_a in the matrix $\mathbf{M}_P^A$ indicates the number of rules headed by auxiliary predicates, we reduce the cosine similarity of each 2-combinations of all rows in the matrix $\mathbf{M}_P^A[k,\cdot,\cdot]$

for generating more possible formats of rules headed by the auxiliary predicates. Then, we calculate the loss function L_F:

$$L_F = \sum_{k=1}^{m_2} \sum_{(i_1,i_2)\in\binom{[1,n_a]}{2}} \text{MSE}(\cos(\mathbf{M}_P^A[k,i_1,\cdot],\mathbf{M}_P^A[k,i_2,\cdot]),-1),$$

where $\binom{[1,n_a]}{2}$ is the set of all 2-combinations of the integer set $[1,n_a]$. To apply the curriculum learning: The system consolidates what it learns in one episode, storing it as background knowledge, and reusing it in subsequent episodes [Evans *et al.*, 2021]. We devise a strategy to implement the curriculum learning that DFOL uses sound logic rules extracted in every few epochs as the prior knowledge to reduce the search space in the following epochs. We reduce the cosine similarity between the rows in the trainable matrix $\mathbf{M}_P$ and a learned matrix $\mathbf{M}_{\text{prior}} \in [0,1]^{m_p \times C}$ corresponding to the sound rules described in Section 3.3, where m_p is the number of extracted sound rules. The loss function L_C is defined as follows:

$$L_C = \sum_{(k_1,k_2)\in[1,m]\times[1,m_p]} \text{MSE}(\cos(\mathbf{M}_P[k_1,\cdot],\mathbf{M}_{\text{prior}}[k_2,\cdot]),-1).$$

In summary, the final loss is the weighted sum of the losses in $\mathbf{L}=[L_I, L_S, L_B, L_O, L_F, L_C]$, i.e., loss $= \Theta \cdot \mathbf{L}$, where Θ is a hyperparameter vector. We use the Adam algorithm [Kingma and Ba, 2015] to minimize the final loss.

3.3 Rule Extraction

In this section, we describe how to extract LPs from a trained SH matrix and the definition of the precision of a rule.

In a trained SH matrix $\mathbf{M}_P$, the element at the m-th row and n-th column represents the possibility of n-th valid body feature in the m-th rule r_m in P. We use multiple thresholds called rule filters, denoted as τ_f, on $\mathbf{M}_P$ to extract rules. Let rule filters range from 0 to 1 with step 0.05, and let T be the set with all rule filters. For a τ_f, we let the valid features in the k-th row of $\mathbf{M}_P$ with values greater than τ_f be the elements in the $body(r_k)$. We iteratively apply each τ_f on the trained matrix $\mathbf{M}_P$, and a rule set $\tilde{R}$ with $m\times|\mathsf{T}|$ rules is generated. Next, we describe the definition of the precision of a rule. Let n_r be the number of the substitutions that satisfy both the body and the head atom of a rule, and let n_b be the number of the substitutions that satisfy only the body of a rule, where the substitutions are computed based on the Datalog and seen facts. Then, we regard the ratio $\frac{n_r}{n_b}$ as the precision of the rule r. A rule is correct with the precision value 1, and a rule is incorrect with the precision value 0. If a precision value floats within the interval $(0,1)$, then the rule may also be correct due to the incompleteness of the seen facts. Therefore, we set another threshold called soundness filter τ_s. Rules with precision values no lower than τ_s are called sound rules and are added to P from the set $\tilde{R}$. To use the curriculum learning strategy, we concentrate the trained rows in $\mathbf{M}_P$ corresponding to the sound rules into $\mathbf{M}_{\text{prior}}$ after every few epochs. The matrix $\mathbf{M}_{\text{prior}}$ is considered as the prior knowledge to boost the training process in the following training epochs. When all training epochs are finished, the sound LP is stored in P.

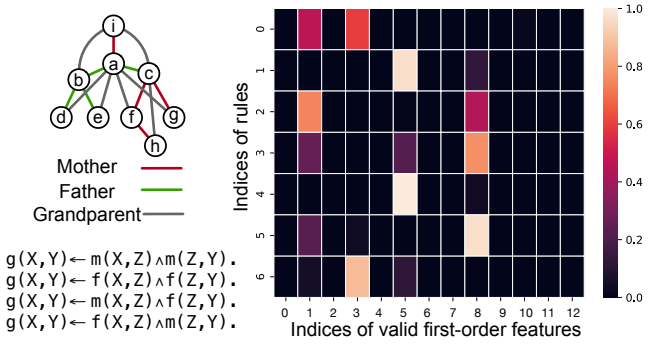

Figure 2: The training facts (at top-left), the learned matrix (at right), and the extracted LP P (at bottom-left) in the *grandparent* task. The valid first-order features with indices 0-12 are $m(X,Y)$, $m(X,Z)$, $m(Y,Z)$, $m(Z,Y)$, $f(X,Y)$, $f(X,Z)$, $f(Y,Z)$, $f(Z,X)$, $f(Z,Y)$, $g(X,Z)$, $g(Y,Z)$, $g(Z,X)$, $g(Z,Y)$.

4 Experimental Evaluation

In this section, we present the performance of DFOL and make comparisons with NTPλ [Rocktäschel and Riedel, 2017], ∂ILP [Evans and Grefenstette, 2018], and NeuralLP [Yang *et al.*, 2017]. The ratio of positive test examples covered by an LP P to all positive test examples is regarded as the accuracy of P, which is also the recall value of P in the test dataset. Due to the small size of trainable matrices in DFOL and to demonstrate the efficiency of DFOL, all experiments are executed on 24GB of memory and an 8-core Intel i7-6700 CPU. We limited the running time to one hour when generating an SHLP. Besides, the variable depth in each task does not exceed two.

4.1 Learning from ILP Datasets

In this section, we present the results of DFOL on 20 classifications of ILP datasets [Evans and Grefenstette, 2018]. Most of the ILP datasets have small numbers of data. In these benchmarks, we set $\tau_s = 1$. Hence, the precision value of each generated rule is 1. The accuracy values of all the generated LPs are 100% in this section. The task definitions and generated solutions are shown in Appendix C. We use the *grandparent* (g) relation as an example to show the results. The background assumptions include the facts with *mother* (m) and *father* (f) relations, and the positive examples include the facts with g relation. The training facts, the learned matrix $\mathbf{M}_P$, and the extracted LP P with 100% accuracy are presented in Figure 2.

Based on the comparison with ∂ILP and NeuralLP, DFOL completes 20 of the 22 benchmarks and outperforms NeuralLP in 19 of all benchmarks. Since ∂ILP is a memory expensive model [Evans and Grefenstette, 2018], ∂ILP cannot generate correct LPs in the case of a large number of objects, e.g., in the even number (20), graph coloring (10) tasks, and knowledge base dataset considered in the paper. Besides, ∂ILP requires a logical template, a more specific prior knowledge than the forward chain format. Hence, DFOL is a precise rule learner for smaller ILP datasets. However, DFOL runs over the memory limit when producing all substitutions in the propositionalization on the *husband* and *uncle* tasks.

	Lt	*Pre*	*Member*	*Son*	*Con*	*DE*
σ	3	3	3	3	2	3
μ	0.95	0.95	0.90	0.95	0.95	0.95

Table 1: The results on ambiguous datasets. The notations *Con* and *DE* represent *connectedness* and *directed edge* tasks, respectively.

4.2 Learning from Ambiguous Datasets

In this section, we test the robustness of DFOL and consider two cases: (1) learning from probabilistic facts; (2) learning from mislabeled facts. We set the soundness filter τ_s to 1 to obtain rules with precision values of 1.

For the first case, each ground atom in the training fact set F and training negative example set $\mathcal{N}$ has a probability. Let $\epsilon \sim N(0, \sigma^2)$, then the probability of fact in F satisfies the distribution $p_i^+ = min(1-\epsilon, 1)$, and the probability of ground atom in $\mathcal{N}$ satisfies the distribution $p_i^- = max(\epsilon, 0)$. The standard deviation σ ranges from 0.5 to 3 with step 0.5. We present the distribution of the positive and negative examples on *lessthan* (*lt*) and *pre* tasks when $\sigma = 3$ in Figures 4 and 5 of Appendix B. From the two distributions, we derive that the values of examples are sufficiently ambiguous when $\sigma = 3$. For the second case, both positive and negative training examples are mislabeled with the mutation rate μ ranging from 0.05 to 1 with step 0.05. In Table 1, we show the maximum standard deviation σ and mutation rate μ when the accuracy values of the generated LPs are 100%. From the results, we conclude that DFOL is robust when handling ambiguous data. When $\mu = 1$, the semantics of head predicate in *lt* task is *largethan* relation, and the result is:

$$lt(X,Y) \leftarrow succ(Y,X),\ lt(X,Y) \leftarrow lt(X,Z) \wedge lt(Z,Y).$$

4.3 Learning from Knowledge Bases

In this section, we test DFOL on three larger real knowledge bases, including Countries [Bouchard *et al.*, 2015], Unified Medical Language System (UMLS), and Nations datasets [Kok and Domingos, 2007]. Let $\tau_s = 0.3$ because the training facts are incomplete in realistic scenarios. We present each rule with the tuple $(\frac{n_r}{n_b}, n_r, n_b)$ in Appendix C. The descriptions of these datasets are presented in Table 3 of Appendix B. For the Countries dataset, training facts are split into S1, S2, and S3 sub-datasets. From the sub-datasets S1 to S3, the learning difficulty is increasing because the related relations are missing corresponding to the test cities [Rocktäschel and Riedel, 2017]. For the Nations and UMLS datasets, we divide each dataset into 80% training facts, 10% development facts, and 10% test facts. We compare LPs generated by DFOL, NTPλ, and NeuralLP, through the indicators of accuracy, mean reciprocal rank (MRR), and HITS@m [Bordes *et al.*, 2013] in Table 2.

From the results in Table 2, we conclude that DFOL has better performance than other baselines in general. Thus, we show that DFOL guarantees the accuracy of the generated LPs, as well as the scalability and interpretability on larger relational datasets. In addition, in all sub-datasets of the Countries dataset, although facts with *locatedIn* and *neighbor* predicates related to the test countries are missing, these facts are still kept in the training facts that are not related to the test countries. Hence, we can extract the same LP from all tasks. The generated symbolic LP can describe all the test facts in both S1 and S2 sub-datasets. However, because of the missing related facts in the S3, the 100% accuracy logic rule needs at least four variables. DFOL runs out of memory when generating all substitutions for four variables in the propositionalization process on the S3 sub-dataset.

Dataset	Metrics	NTPλ	NeuralLP	DFOL
Countries	ACC@S1	**100.00**	**100.00**	**100.00**
	ACC@S2	**100.00**	**100.00**	**100.00**
	ACC@S3	**100.00**	–	–
Nations	MRR	41.79	56.49	**78.88**
	HITS@1	41.79	52.49	**73.88**
	HITS@3	41.79	60.95	**84.58**
	HITS@10	41.79	61.19	**85.07**
	ACC@*blo*	**100.00**	50.00	**100.00**
	ACC@*int*	**84.62**	**84.62**	**84.62**
	ACC@*neg*	37.50	**75.00**	**75.00**
UMLS	MRR	30.03	66.69	**74.96**
	HITS@1	29.95	61.27	**71.41**
	HITS@3	30.11	72.31	**78.82**
	HITS@10	30.11	72.31	**78.97**
	ACC@*isa*	65.96	63.83	**91.48**
	ACC@*intw*	83.67	86.67	**100.00**

Table 2: Comparison on knowledge bases. The results in bold indicate the highest accuracy on the corresponding test datasets. The ACC@Sn represent the accuracy of the generated LP on the Sn subset of Countries. ACC@h represent the accuracy of LP with the head predicate h. *Blo*, *int*, *neg*, and *intw* denote the relations of *blockpositionindex*, *intergovorgs3*, *negativecomm*, and *interacts_with*.

5 Conclusion and Future Work

In this paper, we proposed differentiable first-order rule learner (DFOL), which learns first-order logic programs in the forward-chained format from relational facts without logic templates. Through the proposed propositionalization method, DFOL translates relational data to the neural network-readable data. By applying the proposed constraint functions, DFOL can learn correct first-order logic programs with a few trainable parameters. We extract symbolic logic programs from the trainable matrices directly and apply prior knowledge as constraints to implement curriculum learning. Experimental results indicate that DFOL can learn first-order logic programs with both high accuracy and precision from several standard inductive logic programming tasks, probabilistic relational facts, and knowledge bases. Hence, DFOL can learn rules from relational facts in a flexible, precise, robust, scalable, and computationally cheap manner.

In the future, we will try a more efficient propositionalization method in order to learn relational tasks with larger entities. Besides, we will extend DFOL to an explainable deep neural network framework to generate logic programs with more variables. We also plan to support the function in the term and the negation in the body of a rule.

References

[Bordes *et al.*, 2013] Antoine Bordes, Nicolas Usunier, Alberto García-Durán, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, pages 2787–2795, 2013.

[Bouchard *et al.*, 2015] Guillaume Bouchard, Sameer Singh, and Théo Trouillon. On approximate reasoning capabilities of low-rank vector spaces. In *AAAI*, 2015.

[Cropper *et al.*, 2020] Andrew Cropper, Sebastijan Dumancic, and Stephen H. Muggleton. Turning 30: New ideas in inductive logic programming. In *IJCAI*, pages 4833–4839, 2020.

[d'Avila Garcez and Zaverucha, 1999] Artur S. d'Avila Garcez and Gerson Zaverucha. The connectionist inductive learning and logic programming system. *Appl. Intell.*, 11(1):59–77, 1999.

[d'Avila Garcez *et al.*, 2001] Artur S. d'Avila Garcez, Krysia Broda, and Dov M. Gabbay. Symbolic knowledge extraction from trained neural networks: A sound approach. *Artif. Intell.*, 125(1-2):155–207, 2001.

[Dong *et al.*, 2019] Honghua Dong, Jiayuan Mao, Tian Lin, Chong Wang, Lihong Li, and Denny Zhou. Neural logic machines. In *ICLR*, 2019.

[Evans and Grefenstette, 2018] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *J. Artif. Intell. Res.*, 61:1–64, 2018.

[Evans *et al.*, 2021] Richard Evans, José Hernández-Orallo, Johannes Welbl, Pushmeet Kohli, and Marek J. Sergot. Making sense of sensory input. *Artif. Intell.*, 293:103438, 2021.

[França *et al.*, 2014] Manoel V M França, Gerson Zaverucha, and Artur S d'Avila Garcez. Fast relational learning using bottom clause propositionalization with artificial neural networks. *Mach. Learn.*, 94(1):81–104, 2014.

[Gao *et al.*, 2021] Kun Gao, Hanpin Wang, Yongzhi Cao, and Katsumi Inoue. Learning from interpretation transition using differentiable logic programming semantics. *Mach. Learn.*, 2021.

[Hájek, 1998] Petr Hájek. *Metamathematics of fuzzy logic*, volume 4 of *Trends in Logic*. Kluwer, 1998.

[Hohenecker and Lukasiewicz, 2020] Patrick Hohenecker and Thomas Lukasiewicz. Ontology reasoning with deep neural networks. *J. Artif. Intell. Res.*, 68:503–540, 2020.

[Inoue *et al.*, 2014] Katsumi Inoue, Tony Ribeiro, and Chiaki Sakama. Learning from interpretation transition. *Mach. Learn.*, 94(1):51–79, 2014.

[Kaminski *et al.*, 2018] Tobias Kaminski, Thomas Eiter, and Katsumi Inoue. Exploiting answer set programming with external sources for meta-interpretive learning. *Theory Pract. Log. Program.*, 18(3-4):571–588, 2018.

[Kaur *et al.*, 2019] Navdeep Kaur, Gautam Kunapuli, Saket Joshi, Kristian Kersting, and Sriraam Natarajan. Neural networks for relational data. In *ILP*, volume 11770, pages 62–71, 2019.

[Kingma and Ba, 2015] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Kok and Domingos, 2007] Stanley Kok and Pedro M. Domingos. Statistical predicate invention. In *ICML*, volume 227, pages 433–440, 2007.

[Kramer *et al.*, 2001] Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization approaches to relational data mining*, pages 262–291. Springer, Berlin: Heidelberg, 2001.

[Lehmann *et al.*, 2010] Jens Lehmann, Sebastian Bader, and Pascal Hitzler. Extracting reduced logic programs from artificial neural networks. *Appl. Intell.*, 32(3):249–266, 2010.

[Muggleton *et al.*, 2012] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. ILP turns 20. *Mach. Learn.*, 86(1):3–23, 2012.

[Muggleton *et al.*, 2015] Stephen H Muggleton, Dianhuan Lin, and Alireza Tamaddoni-Nezhad. Meta-interpretive learning of higher-order dyadic datalog: Predicate invention revisited. *Mach. Learn.*, 100(1):49–73, 2015.

[Muggleton, 1991] Stephen Muggleton. Inductive logic programming. *New Gener. Comput.*, 8(4):295–318, 1991.

[Qu *et al.*, 2021] Meng Qu, Junkun Chen, Louis-Pascal Xhonneux, Yoshua Bengio, and Jian Tang. RNNLogic: Learning logic rules for reasoning on knowledge graphs. In *ICLR*, 2021.

[Rocktäschel and Riedel, 2017] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, pages 3788–3800, 2017.

[Sakama *et al.*, 2021] Chiaki Sakama, Katsumi Inoue, and Taisuke Sato. Logic programming in tensor spaces. *Ann. Math. Artif. Intell.*, 89(12):1133–1153, 2021.

[Serafini and d'Avila Garcez, 2016] Luciano Serafini and Artur S. d'Avila Garcez. Logic tensor networks: deep learning and logical reasoning from data and knowledge. In *NeSy@HLAI*, volume 1768, 2016.

[Sourek *et al.*, 2018] Gustav Sourek, Vojtech Aschenbrenner, Filip Zelezný, Steven Schockaert, and Ondrej Kuzelka. Lifted relational neural networks: Efficient learning of latent relational structures. *J. Artif. Intell. Res.*, 62:69–100, 2018.

[Teru *et al.*, 2020] Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *ICML*, pages 9448–9457, 2020.

[Van Emden and Kowalski, 1976] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *J. ACM*, 23(4):733–742, 1976.

[Yang *et al.*, 2017] Fan Yang, Zhilin Yang, and William W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, pages 2319–2328, 2017.