LEARN TO EXPLAIN EFFICIENTLY VIA NEURAL LOGIC INDUCTIVE LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The capability of making interpretable and self-explanatory decisions is essential for developing responsible machine learning systems. In this work, we study the learning to explain problem in the scope of inductive logic programming (ILP). We propose Neural Logic Inductive Learning (NLIL), an efficient differentiable ILP framework that learns first-order logic rules that can explain the patterns in the data. In experiments, compared with the state-of-the-art methods, we find NLIL can search for rules that are x10 times longer while remaining x3 times faster. We also show that NLIL can scale to large image datasets, i.e. Visual Genome, with 1M entities.

1 INTRODUCTION

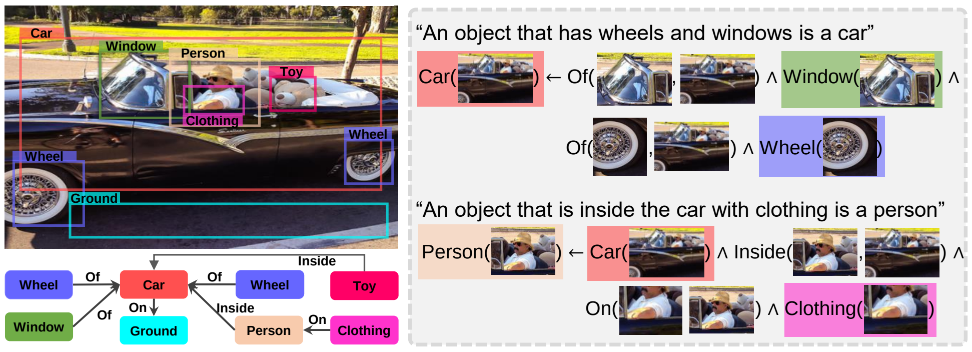

Car
Window
Person
Toy
Clothing
Wheel
Wheel
Ground
Inside
Wheel
Of
Car
Of
Wheel
Toy
On
Of
Inside
On
Window
Ground
Person
Clothing
"An object that has wheels and windows is a car"
Car( ) ← Of(,) ∧ Window( ) ∧
Of(,) ∧ Wheel( )
"An object that is inside the car with clothing is a person"
Person( ) ← Car( ) ∧ Inside(,) ∧
On(,) ∧ Clothing( )

Figure 1: NLIL learns the first-order logic rules as the explanations to the presence of objects `Car` and `Person`.

The recent years have witnessed the growing success of deep learning models in a wide range of applications. However, these models are also criticized for the lack of interpretability in its behavior and decision making process (Lipton, 2016; Mittelstadt et al., 2019), and for being data-hungry. The ability to explain its decision is essential for developing a responsible and robust decision system (Guidotti et al., 2019). On the other hand, logic programming methods, in the form of first-order logic (FOL), are capable of discovering and representing knowledge in explicit symbolic structure that can be understood and examined by human (Evans & Grefenstette, 2018).

In this paper, we investigate the learning to explain problem in the scope of inductive logic programming (ILP) which seeks to learn first-order logic rules that explain the data. Traditional ILP methods (Galárraga et al., 2015) relies on hard matching and discrete logic for rule search which is not tolerant for ambiguous and noisy data (Evans & Grefenstette, 2018). A number of works are proposed for developing differentiable ILP models that combine the strength of neural and logic-based computation (Yang et al., 2017; Evans & Grefenstette, 2018; Campero et al., 2018; Rocktäschel & Riedel, 2017; Payani & Fekri, 2019). Methods such as ∂ILP (Evans & Grefenstette, 2018) are referred to as forward-chaining methods. It constructs rules using a set of pre-defined templates and evaluates them by applying the rule on *background data* multiple times to deduce new facts that lie in the held-out set. Other methods such as NerualLP (Yang et al., 2017) are called backward-chaining methods. Upon given a query, it searches for the rule that, starting from the query, can derive backward towards the known facts in the background set (related works available at Appendix A).

However, general ILP involves several steps that are NP-hard: (i) the rule search space grows exponentially in the formula length; (ii) assigning the logic variables to be shared by predicates grows exponentially in the number of arguments, which we refer as variable binding problem; (iii) the number of rule *instantiations* needed for formula evaluation grows exponentially in the size of data. To alleviate these complexities, most works have limited the search length to within 3 and resort to template-based or chain-like variable assignments. Limiting the expressiveness of the learned rules (detailed discussion available at Appendix B). Still, most of the works are limited in small scale problems with less than 10 relations and 1K entities, with NeuralLP the only exception.

In this work, we propose Neural Logic Inductive Learning (NLIL), a differentiable ILP framework that is highly scalable compared to previous works. We propose a divide-and-conquer strategy and decompose the search space into 3 subspaces in a hierarchy, where each of them can be searched efficiently using attentions. This enables us to search for x10 times longer rules while remaining x3 times faster than the state-of-the-art methods.

And more importantly, we show that a scalable ILP method is widely applicable for model explanations in supervised learning scenario. We apply NLIL on Visual Genome (Krishna et al., 2016) dataset for learning explanations for 150 object classes over 1M entities. We demonstrate that the learned rules while maintaining the interpretability, have comparable predictive power as densely supervised models, and generalize well with less than 1% of the data.

2 PRELIMINARIES

Supervised learning typically involves learning classifiers that map object from its input space to a score between 0 and 1. How can one explain the outcome of a classifier? Recent works on interpretability focus on generating heatmaps or attention that self-explains a classifier (Ribeiro et al., 2016; Chen et al., 2018; Olah et al., 2018). We argue that a more effective and human-intelligent explanation is through the description of the connection with other classifiers.

For example, consider an object detector with classifiers $\texttt{Person}(X)$, $\texttt{Car}(X)$, $\texttt{Clothing}(X)$ and $\texttt{Inside}(X, Y)$ that detect if certain region contains a person, a car, a clothing or is inside another region, respectively. To explain why a person is present, the one can leverage its connection with other attributes, such as "X is a person if it's inside a car and wearing clothing". This intuition draws a close connection to a longstanding problem of first-order logic literature, i.e. **Inductive Logic Programming** (ILP).

2.1 INDUCTIVE LOGIC PROGRAMMING

A typical first-order logic system consists of 3 components: **entity**, **predicate** and **formula**. Entities are objects $\mathbf{x} \in \mathcal{X}$. For example, for a given image, a certain region is an entity $\mathbf{x}$, and the set of all possible regions is $\mathcal{X}$. Predicates are functions that map entities to 0 or 1, for example $\texttt{Person} : \mathbf{x} \mapsto \{0, 1\}$, $\mathbf{x} \in \mathcal{X}$. Classifiers can be seen as *soft* predicates. Predicates can take multiple arguments, e.g. $\texttt{Inside}$ is a predicate with 2 inputs. The number of arguments is referred as the *arity*. **Atom** is a predicate symbol applied to a logic variable, e.g. $\texttt{Person}(X)$ and $\texttt{Inside}(X, Y)$.

A first-order logic (FOL) formula is a combination of atoms using logical operations $\{\wedge, \vee, \neg\}$ which corresponds to logic `and`, `or` and `not` respectively. First-order logic formula is used to express *lifted* knowledge that does not depend on specific data, for example, given a set of predicates $\mathcal{P} = \{P_1 ... P_K\}$, we define the explanation of a predicate P_k as a first-order logic entailment

$$\forall \mathbf{x} \exists \mathbf{y}_1 \mathbf{y}_2 ... \; P_k(\mathbf{x}) \leftarrow A(\mathbf{x}, \mathbf{y}_1, \mathbf{y}_2 ...), \tag{1}$$

where A is called the *body* (and $P_k(\mathbf{x})$ the *head*) of the entailment. A is a general formula, e.g. conjunction normal form (CNF), consisting of atoms with predicate symbols from $\mathcal{P}$ and logic variables that are either head variable $\mathbf{x}$ or one of the body variables $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, ...\}$. These variables make the explanation highly transferrable as they are "lifted" from actual data. Explanations represented as FOL entailments can be easily interpreted. For example,

$$\texttt{Person}(X) \leftarrow \texttt{Inside}(X, Y) \wedge \texttt{Car}(Y) \wedge \texttt{Inside}(Z, X) \wedge \texttt{Clothing}(Z) \tag{2}$$

represents the knowledge that "if an object is inside the car with clothing on it, then it's a person".

Given the set of predicates $\mathcal{P}$ and a set of facts associated to them (usually in the form of a relational knowledge base (KB)). The process of learning FOL rule in the form of Eq.(1) that entails target predicate P^* is called inductive logic programming. For simplicity, we consider unary and binary predicates for the following contents, but this definition can be easily extended to predicates with higher arity.

3 Neural Logic Inductive Learning

3.1 The operator view

We introduce the operator view of the predicate in a logic entailment and show how this can be combined with the attention mechanism to significantly reduce the size of the search space for assigning logic variables.

By definition, variables that only appears in the body are under existential quantifier. We can turn Eq.(1) into *Skolem normal form* by replacing all variables under existential quantifier with functions w.r.t $\mathbf{x}$,

$$\forall \mathbf{x} \exists \varphi_1, \varphi_2, ... \; P^*(\mathbf{x}) \leftarrow A(\mathbf{x}, \varphi_1(\mathbf{x}), \varphi_2(\mathbf{x}), ...). \tag{3}$$

If the functions are already given, Eq.(3) will be much easier to evaluate than Eq.(1). Given a FOL formula, for each instance of variables $\mathbf{x}$ (i.e. the logic variables of a single predicate P^*) we only need to evaluate exactly one grounding formula, because the substitution of the rest of the logic variables are determined by the deterministic functions w.r.t $\mathbf{x}$. Solving the complexity in variable binding and evaluation both at once.

Functions in Eq.(3) are defined as any arbitrary function. But what is the set of functions that one can utilize? We propose to turn each predicate in $\mathcal{P}$ into a *operator*, such that we have a subspace of the functions $\Phi = \{\varphi_1, ..., \varphi_K\}$. Formally, for each predicate P_k we define its operator as a mapping $\varphi_{w_k} : \mathbf{x} \mapsto \mathcal{X}_k$ parameterized by w_k, such that for all instantiations $\mathbf{x} \in \mathcal{X}$, we have

$$\exists \mathbf{y} \in \varphi_{w_k}(\mathbf{x}) \quad P_k(\mathbf{x}, \mathbf{y}) = 1.$$

Intuitively, given a subject entity, a predicate's operator returns the set of object entities that, together with the subject, satisfy the relation. For example, given an image, $\varphi_{\texttt{Inside}}(\mathbf{x})$ which is the operator of binary predicate $\texttt{Inside}(X, Y)$ takes an input entity, i.e. a bounding box, and returns a set of bounding boxes that spatially contains the input box. For unary predicate such as $\texttt{Car}(X)$, its operator $\varphi_{\texttt{Car}}()$ does not depend on the input and simply returns the set of all bounding boxes that contain a car. Operators can take the output of another operator. For example $\varphi_{\texttt{Inside}}(\varphi_{\texttt{Person}}())$ returns the bounding boxes that spatially contains a person. Additionally, we introduce auxiliary operators such as $\varphi_{\texttt{identity}}(\mathbf{x})$ which returns the input argument. This is useful for body formula to re-use the head variables.

By using the operators, we have created a space that represents existential variables $\hat{\mathcal{Y}}$. This approach implies that all variables in $\hat{\mathcal{Y}}$ must be expressed as a sequence of transformations starting from $\mathbf{x}$ only through functions in Φ. Thus $\hat{\mathcal{Y}} \subseteq \mathcal{Y}$. This is slightly constrained from the original definition, but we argue that formulas that contain $\mathbf{y} \notin \hat{\mathcal{Y}}$ are typical of less interest. For example, in $\texttt{Person}(X) \leftarrow \texttt{Car}(Y)$, Y can not be represented by functions of X, thus is a completely free variable. This formula is trivial since it's not likely to infer "an image contains a person" simply by checking if "any car is present in the image".

Solving the variable binding problem can be framed as searching the appropriate chain of operator calls on the head variables. Any FOL formula that complies with Eq.(3) can be converted into operator form and vice versa. For example, Eq.(2) can be written as

$$\texttt{Person}(X) \leftarrow \texttt{Car}(\varphi_{\texttt{Inside}}(X)) \wedge \texttt{Inside}(\varphi_{\texttt{Clothing}}(), \varphi_{\texttt{Identity}}(X)), \tag{4}$$

where the variable Y and Z are eliminated. With K operators, it can effectively represent $(n_h)^K$ number of variables for one step of operator call, where n_h is the arity of the head predicate which is 1 or 2. Some variables may need more than one operator calls to represent. For example, for friendship relation $\texttt{Friend}(X, Y)$, "the friends of the friends of a person" can be represented by stacking the operator two times $\varphi_{\texttt{Friend}}(\varphi_{\texttt{Friend}}(\mathbf{x}))$. Thus, we extend the search space hierarchically by stacking operator calls on previous outputs.

3.2 Primitive Statements

We have now obtained a set of variables $\hat{\mathcal{Y}}$ via operator calls. Now we determine how these variables are assigned a predicate. Note that an *atom* is defined as a predicate symbol applied to specific logic variables. Similarly, we define a predicate applied to the variables obtained from operator calls as a **primitive statement**. For example, in Eq.(4), $\texttt{Car}(\varphi_{\texttt{Inside}}(X))$ and $\texttt{Inside}(\varphi_{\texttt{Clothing}}(), \varphi_{\texttt{Identity}}(X))$ are two primitive statements.

Similar to an atom, each primitive statement is a mapping from the input space to a scalar value between 0 and 1 in soft logic. As suggested by its name, It evaluates the likelihood of a basic statement being true or not. This is equivalent to evaluating its FOL rule, e.g. $\texttt{Inside}(X, Y) \wedge \texttt{Car}(Y)$ and $\texttt{Inside}(Z, X) \wedge \texttt{Clothing}(Z)$. This notion serves as a basic unit that maps from entity space to probability. In fact, each statement is itself a complete, yet tiny, FOL rule. We will be using this notion to construct the searching space for more complex and expressive formulas in the next section.

Remarks: The notion of primitive statement draws a close connection between our work and another differentiable ILP methods, i.e. NeuralLP (Yang et al., 2017). NeuralLP solves the ILP by searching over a chain-like rule space $P^*(\mathbf{x}, \mathbf{y}) \leftarrow P_1(\mathbf{x}, \mathbf{z}_1) \wedge P_2(\mathbf{z}_1, \mathbf{z}_2) \wedge ... \wedge P_n(\mathbf{z_{n-1}}, \mathbf{y})$. Rules of this form can be seen as the "unrolled" primitive statement with one argument fixed and the other one transformed with operators, i.e. $P^*(\varphi_{identity}(\mathbf{x}), \varphi_n \cdot ... \cdot \varphi_2 \cdot \varphi_1(\mathbf{x}))$. If one constrains the body in Eq.(3) to contain only one primitive statement, then NLIL's rule space degenerates to the chain-like rule. Therefore, NLIL can be seen as the generalization over the chain-like rule space of NeuralLP, where it searches over multiple chains simultaneously as candidates and builds up more complex formulas with these candidates as basic units. In section 4, we will see more connections in formula evaluations.

3.3 Formula generation

Each primitive statement can be seen as a self-contained FOL rule. One can construct more expressive rules by searching over the logical combinations of these statements, i.e. $\{\wedge, \vee, \neg\}$. The formula search builds hierarchically upon the previous search results, where the exponential complexities on variable bindings are already encapsulated by primitive statements, making the search highly efficient.

To do so, we first define the soft logic operations that enable the gradient to flow through. Specifically, we follow the common definition of soft logic `not` and soft logic `and`

$$f_\neg(p) = 1 - p \qquad f_\wedge(p, q) = p * q, \qquad f_s(p; \alpha_0, \alpha_1) = \alpha_0 p + \alpha_1 f_\neg(p)$$

where p and q are probabilities between 0 and 1. And we define a soft select function f_s with α_0, α_1 as the mixing parameters that are taken from an attention vector. With appropriate parameters, f_s can represents p and $\neg p$. We define a general logic operator as

$$f_{logic}(p, q; \alpha) = f_s(f_\wedge(f_s(p), f_s(q))), \tag{5}$$

which can represent p, $\neg p$, $p \vee q$, $p \wedge q$ with proper parameters. However, this can also lead to trivial expressions such as $p \vee \neg p$. This is generally more difficult to recognize than that in primitive statement search. We counter this by introducing negative samples in evaluation phase discussed in section 4.

Searching in the formula space can be carried out hierarchically, which is similar to that in operator search. Given a set of statements, one searches over all their one-step-combinations using Eq.(5). And the next step will explore the combinations of the outputs of the current step.

4 End-to-End Evaluation and Scalable Training

We have discussed how to decompose the formula search into a sequence of hierarchical searches, each one with no more than quadratic complexities. In this section, we introduce the model for learning to search in this space and its training method.

The goal is to learn logic rules that are in first-order, which means the rules generated should be globally consistent and does not change with specific data. Forward-chaining methods such as ∂ILP

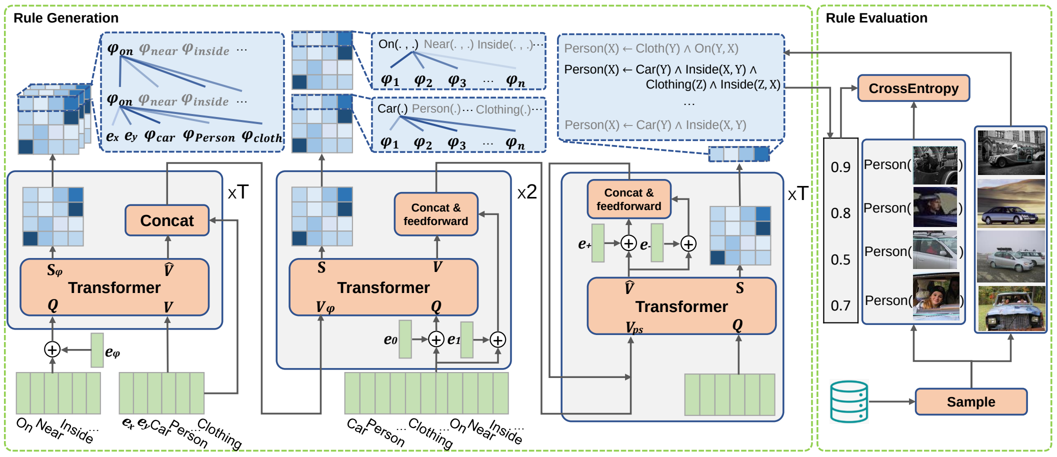

Rule Generation
Rule Evaluation
Concat
Transformer
xT
x2
Concat & feedforward
CrossEntropy
Sample
Person(X) ← Cloth(Y) ∧ On(Y, X)
Person(X) ← Car(Y) ∧ Inside(X, Y) ∧ Clothing(Z) ∧ Inside(Z, X)
Person(X) ← Car(Y) ∧ Inside(X, Y)
0.9
0.8
0.5
0.7
On Near Inside ...
e_x e_y Car Person ... Clothing
Car Person ... Clothing ... On Near Inside ...

Figure 2: The model architecture of Neural Logic Inductive Learning.

achieves this via rule-templates, while NeuralLP, a backward-chaining method, cannot learn FOL rules because it depends on the query. On the other hand, since NeuralLP generates the rule on-the-fly, it's more efficient in rule evaluation compared to those forward-chaining counter-parts.

In NLIL, we propose a hybrid approach that is in first-order and efficient for evaluation. Specifically, as shown in Figure 2, we split the framework into two parts: the rule generation and rule evaluation. The rule generation does not depend on the data. The inputs are predicate and logic variable embeddings. The model is parameterized with a stack of Transformers (Vaswani et al., 2017) that output a sequence of attentions representing the soft picks of rules. The rule evaluation happens after the rules are generated, it samples data from the knowledge base (KB) and computes the cross-entropy loss using the generated rules, which yields gradients that are to be back-propagated through the entire model.

4.1 Hierarchical search via attentions

Operator search: We propose using attention mechanism to efficiently represent and search in this space. Specifically, for all K predicates, we define their learnable embeddings as $H = [\mathbf{h}_1, .., \mathbf{h}_K]$, and H_b is the embeddings for binary predicates alone. Without the loss of generality, suppose the P^* is a binary predicate, then the attentions are generated as

$$\mathbf{V}^{(0)} = [\mathbf{e}_X, \mathbf{e}_Y], \qquad \mathbf{Q} = H_b + \mathbf{e}_\varphi,$$
$$\hat{\mathbf{V}}^{(t)}, \mathbf{S}_\varphi^{(t)} = \text{MultiHeadAttn}(\mathbf{Q}, \mathbf{V}^{(t-1)}), \qquad \mathbf{V}^{(t)} = \text{Concat}(\mathbf{V}^{(t-1)}, \hat{\mathbf{V}}^{(t)}),$$

where $\mathbf{e}_X$, $\mathbf{e}_Y$ and $\mathbf{e}_\varphi$ are learnable embeddings of logic variable X, Y and operator encoding. The MultiHeadAttn is a standard Transformer module that takes the *query* and input value (which will be internally transformed into *keys* and *values*), and returns the soft representation $\hat{\mathbf{V}}^{(t)}$ and attention matrix $\mathbf{S}_\varphi^{(t)}$. Formally, $\text{MultiHeadAttn} : \mathbf{Q}^{q\times d}, \mathbf{V}^{v\times d} \mapsto \hat{\mathbf{V}}^{q\times d}, \mathbf{S}^{q\times v}$

For each step of the operator call search, the operator embeddings of binary predicates are the queries. And head variable embeddings concatenated with all past Transformer outputs are the input value. The output attention $\mathbf{S}_\varphi$ represents the soft choice of an operator over the possible arguments. The notation (t) means we perform the search for (t) times, which enables us to search hierarchically in the variable space.

Primitive statement search: Searching for primitive statements is similar to that in operator space. Lets denote the soft representation of all variables as $\mathbf{V}_\varphi = [\mathbf{V}^{(0)}, ..., \mathbf{V}^{(T)}]$, and assume all predicates are binary. Then the attention is generated as

$$\mathbf{Q}_0 = H + \mathbf{e}_0, \qquad \hat{\mathbf{Q}}_1 = H + \mathbf{e}_1,$$
$$\mathbf{V}_0, \mathbf{S}_0 = \text{MultiHeadAttn}(\mathbf{Q}_0, \mathbf{V}_\varphi), \qquad \mathbf{Q}_1 = \text{FeedForward}(\text{Concat}(\hat{\mathbf{Q}}_1, \mathbf{V}_0)),$$
$$\mathbf{V}_1, \mathbf{S}_1 = \text{MultiHeadAttn}(\mathbf{Q}_1, \mathbf{V}_\varphi), \qquad \mathbf{V}_{ps} = \text{FeedForward}(\text{Concat}(\mathbf{V}_0, \mathbf{V}_1)),$$

where $\mathbf{e}_0$ and $\mathbf{e}_1$ are the learnable argument position encodings. And $\mathbf{S}_0$ and $\mathbf{S}_1$ are the soft choices of predicates' first and second arguments and $\mathbf{V}_{ps}$ is the representation of all primitive statements. Note that the operator of a unary predicate does not take input, thus the search space in fact contains trivial statements such as $\texttt{Inside}(\varphi_{\texttt{Car}}(), \varphi_{\texttt{Person}}())$, i.e. it does not take any of the head variables. We avoid these candidates by masking out the choices in the decoding phase of the Transformer.

Formula search: The logic combination can be also parameterized with attentions

$$\begin{aligned}
&\mathbf{V}^{(0)}, \mathbf{S}_{ps} = \text{MultiHeadAttn}(\mathbf{Q}_{ps}, \mathbf{V}_{ps}), && \hat{\mathbf{V}}^{(t-1)} = [\mathbf{V}^{(t-1)} + \mathbf{e}_{+}, \mathbf{V}^{(t-1)} + \mathbf{e}_{\neg}],\\
&\mathbf{V}_l^{(t)}, \mathbf{S}_l^{(t)} = \text{MultiHeadAttn}(\mathbf{Q}_l^{(t)}, \hat{\mathbf{V}}^{(t-1)}), && \mathbf{Q}_r^{(t)} = \text{FeedForward}(\text{Concat}(\mathbf{Q}_l^{(t)}, \mathbf{V}_l^{(t)}))\\
&\mathbf{V}_r^{(t)}, \mathbf{S}_r^{(t)} = \text{MultiHeadAttn}(\mathbf{Q}_r^{(t)}, \hat{\mathbf{V}}^{(t-1)}), && \mathbf{V}^{(t)} = \text{FeedForward}(\text{Concat}(\mathbf{V}_l^{(t)}, \mathbf{V}_r^{(t)})),
\end{aligned}$$

where $\mathbf{e}_{+}$, $\mathbf{e}_{\neg}$ are learnable embeddings for positive and negative statement encodings. Given the embeddings of primitive statements $\mathbf{V}_{ps}$, we first softly select those of interest with query $\mathbf{Q}_{ps} \in \mathbf{R}^{c^{(0)} \times d}$ which is a set of learnable embeddings. The filtered embeddings are repeated and transformed with logic encoding $\mathbf{e}_{+}$ and $\mathbf{e}_{\neg}$ representing the positive and negative choices. For each level (t), $\mathbf{Q}_l^{(t)} \in \mathbf{R}^{c^{(t)} \times d}$ is the query that softly picks the left operands of Eq.(5), the right query $\mathbf{Q}_r^{(t)}$ depends on the choice of left operands. And finally, the embeddings of left and right operands are combined with a feed forward network, producing the formula embedding $\mathbf{V}^{(t)}$ to be fed to next level.

By stacking multiple Transformer blocks, one can explore formulas starting from simple logic conjunctions and pass the promising ones to the later block to learn more complex ones. After performing T levels, we have $c^{(0)} + 2\sum_{t=1}^{T} c^{(i)}$ formula candidates generated. Each can be converted explicitly into a FOL entailment. Finally, the very last layer decodes over all previous candidates $\mathbf{V} = [\mathbf{V}^{(0)}, ..., \mathbf{V}^{(T)}]$ and generates the attention that softly picks the best FOL rule.

4.2 End-to-end evaluation

Without the loss of generality, assuming all predicates are binary, the first-order logic rules are evaluated over a set of triples $\{\langle \mathbf{x}, P_k, \mathbf{x}' \rangle_i\}_{i=1}^{N}$, i.e. a relational knowledge base (KB) if organized in terms of predicate. We have shown in section 3 that primitive statements are equivalent to a chain-like rule. Thus we could adopt the setting in NerualLP for efficient evaluation. Specifically, each predicate P_k is represented as a binary matrix $\mathbf{M}_{P_k} \in \{0,1\}^{C \times C}$. Therefore, each operator φ_k is parameterized by this matrix. Given an one-hot encoding $\mathbf{v}_\mathbf{x}$, we have $\varphi_k(\mathbf{v}_\mathbf{x}) = \mathbf{M}_{P_k} \cdot \mathbf{v}_\mathbf{x}$. Thus for an arbitrary primitive statement $P_k(\varphi_{n_l} \cdot ... \cdot \varphi_{n_0}(\mathbf{x}), \varphi_{m_r} \cdot ... \cdot \varphi_{m_0}(\mathbf{y}))$, its value is computed as (detailed proofs available at (Yang et al., 2017))

$$\text{score}(\mathbf{x}, \mathbf{y}) = (\mathbf{M}_{P_k} \prod_{i=l}^{0} \mathbf{M}_{P_{n_i}} \mathbf{v}_\mathbf{x})^T (\prod_{j=r}^{0} \mathbf{M}_{P_{m_j}} \mathbf{v}_\mathbf{y}). \tag{6}$$

For target predicate P^*, let's define the value of i-th rule candidate generated at level (t) is $f_i^{(t)}$, then we have the final output value $f^{*(T+1)}$ as

$$\begin{aligned}
&\mathbf{f}^{(0)} = [\underset{1}{\text{score}}(\mathbf{x}, \mathbf{y}), .., \underset{K}{\text{score}}(\mathbf{x}, \mathbf{y})]^T, && \hat{\mathbf{f}}^{(t-1)} = [\mathbf{f}^{(t-1)}, 1 - \mathbf{f}^{(t-1)}],\\
&f_i^{(t)} = {\mathbf{s}_{l,i}^{(t)}}^T \hat{\mathbf{f}}^{(t-1)} * {\mathbf{s}_{r,i}^{(t)}}^T \hat{\mathbf{f}}^{(t-1)}, && f^{*(T+1)} = {\mathbf{s}^{(T+1)}}^T [\mathbf{f}^{(0)}, .., \mathbf{f}^{(T)}].
\end{aligned}$$

Note that the logic combination Eq.(5) are carried out here implicitly. Thus for all *queries* on target predicate $\{\langle \mathbf{x}, P^*, \mathbf{x}' \rangle_i\}_{i=1}^{N}$, we minimize the loss $l = \frac{1}{N}\sum_i^N \text{CrossEntropy}(1, f^{*(T+1)}(\mathbf{x}, \mathbf{x}'))$. That is, for each query, we only ground the formula once for evaluation. Thus we can also perform stochastic training for better scalability.

During rule generation, the picked variables and rules are represented with attentions. For evaluation, one can either make hard samples from attention using Gumbel-softmax (Jang et al., 2016) and *straight-through* for back-propagating the soft gradient. However, we found in the experiments that directly performing weighted average over the inputs constantly outperforms the former. So we only use Gumbel-softmax sampling for testing and visualization.

Table 1: MRR and Hits@10 results. TransE results are taken from (Sun et al., 2019).

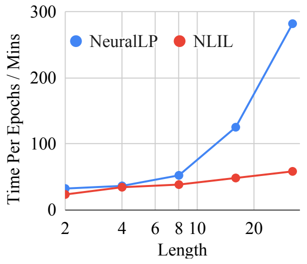
Model	FB15K-237			WN18		
	MRR	Hits@10	Time	MRR	Hits@10	Time
NeuralLP	0.24	36.2	250	0.94	94.5	54
TransE	**0.29**	**46.5**	-	0.50	94.3	-
NLIL	0.25	32.4	**82**	**0.95**	**94.6**	**12**

Table 2: Statistics of benchmark KBs and Visual Genome scene-graphs.

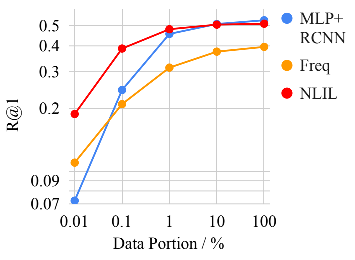
KB	# facts	# entities	# predicates
ES-10	17	10	3
ES-50	77	50	3
ES-1K	1.5K	1K	3
WN18	106K	40K	18
FB15K	272K	15K	237
VG	1.9M	1.4M	2100

Model	ES-10	ES-50	ES-1K
∂ILP	5.6	240	-
NeuralLP	0.1	0.1	0.2
NLIL	**<0.1**	**<0.1**	**0.1**

(a)



(b)



(c)

Figure 3: (a) Time (mins) for solving Even-and-Successor tasks. (-) indicates method runs out of time limit; (b) Running time for different rule lengths; (c) R@1 for object classification with different training set size.

5 Experiments

We first evaluate NLIL on classical ILP benchmarks and compare it with 3 state-of-the-art KB completion methods in terms of their accuracy and efficiency. Then we show NLIL is capable of learning FOL explanations for object classifiers on large image dataset when scene-graphs are present. Though each scene-graph corresponds to a small KB, the total amount of the graphs makes it infeasible for all classical ILP methods. We show that NLIL can overcome it via efficient stochastic training.

5.1 Classical ILP benchmarks

We evaluate NLIL together with two state-of-the-art differentiable ILP methods, i.e. NeuralLP[1] (Yang et al., 2017) and ∂ILP[2] (Evans & Grefenstette, 2018), and an efficient statistical relational learning method, TransE (Bordes et al., 2013). We create separate Transformer blocks for each target predicate, the embedding size for each block is set to 32. All experiments are conducted on a machine with i7-8700K, 32G RAM and one GTX1080ti.

Benchmark datasets: (i) Even-and-Successor (ES) benchmark introduced in (Evans & Grefenstette, 2018), which involves two unary predicate $\texttt{Even}(X)$, $\texttt{Zero}(X)$ and one binary predicate $\texttt{Succ}(X, Y)$. The goal is to learn FOL rules over a set of integers. The benchmark is evaluated with 10, 50 and 1K consecutive integers starting at 0; (ii) FB15K-237 is a subset of the Freebase knowledge base (Toutanova & Chen, 2015) containing general knowledge facts; (iii) WN18 (Bordes et al., 2013) is the subset of WordNet containing relations between words. Statistics of datasets are provided in Table 2.

Knowledge base completion: All models are evaluated on KB completion task. The benchmark datasets are split into train/valid/test sets each containing the fact triplets in the form of $\langle \mathbf{x}, P_k, \mathbf{x}' \rangle$. The model is tasked to predict the probability of a fact triplet (query) being present in the KB. We use Mean Reciprocal Ranks (MRR) and Hits@10 for evaluation metrics (see Appendix C for details).

Results on Even-and-Successor benchmark are shown in Table 3a. Since the benchmark is noise-free, we only show the wall clock time for completely solving the task. As we have previously mentioned, forward-chaining method, i.e. ∂ILP scales exponentially in the number of facts and quickly becomes infeasible for 1K entities. Thus, we skip its evaluation for other benchmarks.

[1]We use the official implementation at https://github.com/fanyangxyz/Neural-LP

[2]We use the third-party implementation at https://github.com/ai-systems/DILP-Core

Results on FB15K-237 and WN18 are shown in Table. 1. All 3 methods achieve similar performance on both benchmarks, with TransE slightly outperforms on FB15K-237 and NLIL on WN18. NLIL and NeuralLP yield similar scores. This is due to the benchmarks favor symmetric/asymmetric relations or compositions of a few relations (Sun et al., 2019), most valuable rules will already lie within the chain-like search space of NeuralLP. Thus the improvements gained from a larger search space with NLIL are limited. On the other hand, with the Transformer block and smaller model created for each target predicate, NLIL can achieve a similar score at least 3 times faster.

Scalability for long rules: we demonstrate that NLIL can explore longer rules efficiently. We compare the wall clock time of NeuralLP and NLIL for performing one epoch of training against different maximum rule lengths. As shown in Figure 3b, NeuralLP searches over a chain-like rule space thus scales linearly with the length, while NLIL searches over a hierarchical space thus grows in log scale. The search time for length 32 in NLIL is similar to that for length 3 in NerualLP.

5.2 ILP ON VISUAL GENOME DATASET

The ability to perform ILP efficiently extends the applications of NLIL to beyond canonical KB completion. For example in visual object detection and relation learning, supervised models can learn to generate a *scene-graph* (As in Figure 1) for each image. It consists of nodes each labeled as an object class. And each pair of objects are connected with one type of relation. The scene-graph can then be, again, represented as a relational KB which one can perform ILP over. Learning the FOL rules on such output of a supervised model is beneficial. As it provides an alternative way of interpreting model behaviors in terms of its relations with other classifiers that is consistent across the dataset.

Table 3: R@1 and R@5 for 150 objects classification on VG.

Model	Visual Genome	
	R@1	R@5
MLP+RCNN	**0.53**	**0.81**
Freq	0.40	0.44
NLIL	0.51	0.52

To show this, we conduct experiments on Visual Genome dataset (Krishna et al., 2016). The original dataset is highly noisy (Zellers et al., 2018), so we use a pre-processed version available as the GQA dataset (Hudson & Manning, 2019). The scene-graphs are converted to a collection KBs, and its statistics are shown in Table 2. We filter out the predicates with less than 1500 occurrences. The processed KBs contain 213 predicates. Then we perform ILP on learning the explanations for the top 150 objects in the dataset.

Quantitatively, we evaluate the learned rules on predicting the object class labels on a held-out set in terms of their R@1 and R@5. As none of the ILP works scale to this benchmark, we compare NLIL with two supervised baselines: (i) **MLP-RCNN**: a MLP classifier with RCNN features of the object (available in GQA dataset) as input; and (ii) **Freq**: a frequency-based baseline that predicts object label by looking at the mostly occurred object class in the relation that contains the target. This method is nontrivial. As noted in (Zellers et al., 2018), a large number of triples in Visual Genome are highly predictive by knowing only the relation type and either one of the object or subject.

Explaining objects with rules: Results are shown in Table 3. We see that the supervised method achieves the best scores, as it relies on highly informative visual features. On the other hand, NLIL achieves a comparable score on R@1 solely relying on KBs with sparse binary labels. We note that NLIL outperforms **Freq** significantly. This means the FOL rules learned by NLIL are beyond the superficial correlations exhibited by the dataset. We verify this finding by showing the rules for top objects in Table 4.

Induction for few-shot learning: Logic inductive learning is data-efficient and the learned rules are highly transferrable. To see this, we vary the size of the training set and compare the R@1 scores for 3 methods. As shown in Figure 3c, the NLIL maintains a achieves similar R@1 score with less than 1% of the training set.

6 CONCLUSION

In this work, we propose Neural Logic Inductive Learning, a differentiable ILP framework that learns explanatory rules from data. We demonstrate that NLIL can scale to very large datasets while being able to search over complex and expressive rules. More importantly, we show that a scalable ILP method is effective in explaining decisions of supervised models, which provides an alternative perspective for inspecting the decision process of machine learning systems.

REFERENCES

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pp. 2787–2795, 2013.

Andres Campero, Aldo Pareja, Tim Klinger, Josh Tenenbaum, and Sebastian Riedel. Logical rule induction and theory learning using neural theorem proving. *arXiv preprint arXiv:1809.02193*, 2018.

Xinlei Chen, Li-Jia Li, Li Fei-Fei, and Abhinav Gupta. Iterative visual reasoning beyond convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7239–7248, 2018.

Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

Luis Galárraga, Christina Teflioudi, Katja Hose, and Fabian M Suchanek. Fast rule mining in ontological knowledge bases with amie+. *The VLDB JournalThe International Journal on Very Large Data Bases*, 24(6):707–730, 2015.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):93, 2019.

Vinh Thinh Ho, Daria Stepanova, Mohamed H Gad-Elrab, Evgeny Kharlamov, and Gerhard Weikum. Rule learning from knowledge graphs guided by embedding models. In *International Semantic Web Conference*, pp. 72–90. Springer, 2018.

Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6700–6709, 2019.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A Shamma, Michael Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. 2016. URL `https://arxiv.org/abs/1602.07332`.

Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pp. 146–160. Springer, 1994.

Zachary C Lipton. The mythos of model interpretability. *arXiv preprint arXiv:1606.03490*, 2016.

Brent Mittelstadt, Chris Russell, and Sandra Wachter. Explaining explanations in ai. In *Proceedings of the conference on fairness, accountability, and transparency*, pp. 279–288. ACM, 2019.

Chris Olah, Arvind Satyanarayan, Ian Johnson, Shan Carter, Ludwig Schubert, Katherine Ye, and Alexander Mordvintsev. The building blocks of interpretability. *Distill*, 3(3):e10, 2018.

Pouya Ghiasnezhad Omran, Kewen Wang, and Zhe Wang. Scalable rule learning via learning representation. In *IJCAI*, pp. 2149–2155, 2018.

Ali Payani and Faramarz Fekri. Inductive logic programming via differentiable deep neural logic networks. *arXiv preprint arXiv:1906.03523*, 2019.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1135–1144. ACM, 2016.

Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pp. 3788–3800, 2017.

Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, pp. 57–66, 2015.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems*, pp. 2319–2328, 2017.

Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural motifs: Scene graph parsing with global context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5831–5840, 2018.

A RELATED WORK

Inductive Logic Programming (ILP) is the task where, given the observed data, it seeks to summarize the underlying patterns shared in the data and express it as a set of logic programs (or rule/formulae) (Lavrac & Dzeroski, 1994). Approaches for solving ILP can be generally grouped as forward-chaining and backward-chaining methods.

Traditional ILP methods such as AMIE+ Galárraga et al. (2015) and RLvLR Omran et al. (2018) relies on explicit search-based method for rule mining with various pruning techniques. These works can scale up to very large knowledge bases. However, the algorithm complexity grows exponentially in the size of the variables and predicates involved. The acquired rules are often restricted to Horn clauses with maximum length less than 3, limiting the expressiveness of the logical rules. On the other hand, compared with differentiable approaches, traditional methods make use of hard matching and discrete logic that lacks the tolerance for ambiguous and noisy data.

The state-of-the-art differentiable forward-chaining methods focus on rule learning on predefined templates (Evans & Grefenstette, 2018; Campero et al., 2018; Ho et al., 2018) (NTP (Rocktäschel & Riedel, 2017) is a backward-chaining method but uses templates as well), typically in the form of a Horn clause with one head predicate and two body predicates with chain-like variables, i.e.

$$P(X,Y) \leftarrow P_1(X,Z) \wedge P_2(Z,Y).$$

To evaluate the rules, one starts with a *background* set of facts and repeatedly apply rules for every possible triple until no new facts can be deduced. Then the deduced facts are compared with a held-out ground-truth set. Rules that are learned in this approach are in first-order, i.e. data-independent and can be readily interpreted. However, the deducing phase can quickly become infeasible with a larger background set. Although ∂ILP (Evans & Grefenstette, 2018) has proposed to alleviate by performing only a fixed number steps, works of this type generally only scale to KBs with less than 1K facts and 100 entities.

Backward-chaining methods such as NeuralLP (Yang et al., 2017) constructs rule on-the-fly when given a specific query. It adopts a flexible ILP setting: instead of pre-defining templates, it assumes a chain like Horn clause can be constructed to answer the query

$$P(X,Y) \leftarrow P_1(X,Z_1) \wedge P_2(Z_1,Z_2) \wedge ... \wedge P_n(Z_{n-1},Y).$$

And each step of the reasoning in the chain can be efficiently represented by matrix multiplication. The resulting algorithm is highly scalable compared to the forward-chaining counter-parts and can learn rules on large datasets such as FreeBase. The main drawback of NeuralLP is that the rule generation dependents on the specific query, i.e. it's data-dependent. Thus making it difficult to extract FOL rules that are interpretable and transferrable. On the other hand, while it can learn rules without templates, the form of the formula is still restricted to chain-like Horn clauses.

Table 4: Example rules learned by NLIL

$\texttt{Person}(X) \leftarrow (\texttt{Shirt}(Y) \wedge \texttt{Wearing}(X,Y)) \vee (\texttt{Pants}(Z) \wedge \texttt{Wearing}(X,Z)) \vee (\texttt{Street}(Z_1) \wedge \texttt{WalkingOn}(X,Z_1))$
$\texttt{Tree}(X) \leftarrow (\texttt{Leaf}(Y) \wedge \texttt{At}(Y,X)) \vee (\texttt{SideWalk}(Z) \wedge \texttt{Near}(Z,X))$
$\texttt{Shirt}(X) \leftarrow (\texttt{Person}(Y) \wedge \texttt{Wearing}(Y,X)) \vee (\texttt{Child}(Z) \wedge \texttt{Wearing}(Z,X))$
$\texttt{Sky}(X) \leftarrow (\texttt{Clouds}(Y) \wedge \texttt{in}(Y,X)) \vee (\texttt{Airplane}(Y) \wedge \texttt{Below}(X,Y))$
$\texttt{Head}(X) \leftarrow \texttt{Helmet}(Y) \wedge \texttt{Above}(Y,X)$
$\texttt{Head}(X) \leftarrow \texttt{Wearing}(Y,Z) \wedge \texttt{SittingOn}(X,Y) \wedge \texttt{Hat}(Z)$
$\texttt{Sign}(X) \leftarrow (\texttt{Number}(Y) \wedge \texttt{On}(Y,X)) \vee (\texttt{Post}(Z) \wedge \texttt{On}(Z,X)) \vee (\texttt{Letter}(Z_1) \wedge \texttt{In}(Z_1,X))$
$\texttt{Sign}(X) \leftarrow \texttt{StreetLight}(Y) \wedge \texttt{On}(Y,Z) \wedge \texttt{On}(X,Z)$
$\texttt{Ground}(X) \leftarrow (\texttt{Dog}(Y) \wedge \texttt{On}(Y,X)) \vee (\texttt{Grass}(Z) \wedge \texttt{CoveredBy}(X,Z))$
$\texttt{Car}(X) \leftarrow \texttt{Wheel}(Y) \wedge \texttt{Of}(Y,X) \wedge \texttt{Window}(Z) \wedge \texttt{Of}(Z,X)$
$\texttt{Sidewalk}(X) \leftarrow \texttt{Person}(Y) \wedge \texttt{WalkingOn}(Y,X) \wedge \texttt{Street}(Z) \wedge \texttt{Near}(X,Z)$
$\texttt{Car}(X) \leftarrow \texttt{Wheel}(Y) \wedge \texttt{Of}(Y,X) \wedge \texttt{Window}(Z) \wedge \texttt{Of}(Z,X)$
$\texttt{Ear}(X) \leftarrow \texttt{Eye}(Y) \wedge \texttt{Of}(Y,Z) \wedge \texttt{Of}(X,Z)$
$\texttt{Chair}(X) \leftarrow \texttt{Arm}(Y) \wedge \texttt{In}(Y,X) \wedge \texttt{Person}(Z) \wedge \texttt{SittingOn}(Z,X)$

B Challenges in ILP

Standard ILP approaches are difficult and involve several procedures that have been proved to be NP-hard. The complexity comes from 3 levels: first, the search space for a formula is vast. The body of the entailment can be arbitrarily long and the same predicate can appear multiple times with different variables, for example, the `Inside` predicate in Eq.(2) appears twice. Most ILP works constrain the logic entailment to be Horn clause, i.e. the body of the entailment is a flat conjunction over literals, and the length limited within 3 for large datasets.

Second, constructing formulas also involves assigning logic variables that are shared across different predicates, which we refer to as **variable binding**. For example, in Eq.(2), to express that a person is inside the car, we use X and Y to represent the region of a person and that of a car, and the same two variables are used in `Inside` to express their relations. Different bindings lead to different meanings. For a formula with n arguments (Eq.(2) has 7), there are $\mathcal{O}(n^n)$ possible assignments. Existing ILP works either resort to constructing formula from pre-defined templates (Evans & Grefenstette, 2018; Campero et al., 2018) or from chain-like variable reference (Yang et al., 2017), limiting the expressiveness of the learned rules.

Finally, evaluating a formula candidate is expensive. A FOL rule is data-independent. To evaluate it, one needs to replace the variables with actual entities and compute its value. This is referred to as *grounding* or *instantiation*. Each variable used in a formula can be grounded independently, meaning a formula with n variables can be instantiated into $\mathcal{O}(C^n)$ grounded formulas, where C is the number of total entities. For example, Eq.(2) contains 3 logic variables: X, Y and Z. To evaluate this formula, one needs to instantiate these variables into C^3 possible combinations, and check if the rule holds or not in each case. However in many domains, such as object detection, such grounding space is vast (e.g. all possible bounding boxes of an image) making the full evaluation infeasible. Many forward-chaining methods such as ∂ILP (Evans & Grefenstette, 2018) scales exponentially in the size of the grounding space, thus are limited to small scale datasets with less than 10 predicates and 1K entities.

C Experiments

Model setting: For KB completion task, we set the number of operator calls to 2 and formula combinations to 1 in NLIL, as most of the relations in those benchmarks can be recovered by symmetric/asymmetric relations or compositions of a few relations (Sun et al., 2019). Thus complex formulas are not preferred. For FB15K-237, binary predicates are grouped hierarchically into domains. To avoid unnecessary search overhead, we use the most frequent 20 predicates that share the same root domain (e.g. “award”, “location”) with the head predicate for rule body construction, which is a similar treatment as in (Yang et al., 2017).

Evaluation metrics: Following the conventions in (Yang et al., 2017; Bordes et al., 2013) we use Mean Reciprocal Ranks (MRR) and Hits@10 for evaluation metrics. For each query $\langle \mathbf{x}, P_k, \mathbf{x}' \rangle$, the model generates a ranking list over all possible groundings of predicate P_k, with other ground-truth triplets filtered out. Then MRR is the average of the reciprocal rank of the queries in their corresponding lists, and Hits@10 is the percentage of queries that are ranked within the top 10 in the list.