Probabilistic Logic Neural Networks for Reasoning

Meng Qu\textsuperscript{1,2}, Jian Tang\textsuperscript{1,3,4}
\textsuperscript{1}Mila - Quebec AI Institute \textsuperscript{2}University of Montréal
\textsuperscript{3}HEC Montréal \textsuperscript{4}CIFAR AI Research Chair

Abstract

Knowledge graph reasoning, which aims at predicting the missing facts through reasoning with the observed facts, is critical to many applications. Such a problem has been widely explored by traditional logic rule-based approaches and recent knowledge graph embedding methods. A principled logic rule-based approach is the Markov Logic Network (MLN), which is able to leverage domain knowledge with first-order logic and meanwhile handle their uncertainty. However, the inference of MLNs is usually very difficult due to the complicated graph structures. Different from MLNs, knowledge graph embedding methods (e.g. TransE, DistMult) learn effective entity and relation embeddings for reasoning, which are much more effective and efficient. However, they are unable to leverage domain knowledge. In this paper, we propose the probabilistic Logic Neural Network (pLogicNet), which combines the advantages of both methods. A pLogicNet defines the joint distribution of all possible triplets by using a Markov logic network with first-order logic, which can be efficiently optimized with the variational EM algorithm. In the E-step, a knowledge graph embedding model is used for inferring the missing triplets, while in the M-step, the weights of logic rules are updated based on both the observed and predicted triplets. Experiments on multiple knowledge graphs prove the effectiveness of pLogicNet over many competitive baselines.

1 Introduction

Many real-world entities are interconnected with each other through various types of relationships, forming massive relational data. Naturally, such relational data can be characterized by a set of \((h,r,t)\) triplets, meaning that entity \(h\) has relation \(r\) with entity \(t\). To store the triplets, many knowledge graphs have been constructed such as Freebase \cite{2} and WordNet \cite{21}. These graphs have been proven useful in many tasks, such as question answering \cite{45}, relation extraction \cite{30} and recommender systems \cite{4}. However, one big challenge of knowledge graphs is that their coverage is limited. Therefore, one fundamental problem is how to predict the missing links based on the existing triplets.

One type of methods for reasoning on knowledge graphs are the symbolic logic rule-based approaches \cite{12,15,31,37,42}. These rules can be either handcrafted by domain experts \cite{38} or mined from knowledge graphs themselves \cite{10}. Traditional methods such as expert systems \cite{12,15} use hard logic rules for prediction. For example, given a logic rule \(\forall x, y, \text{Husband}(x,y) \Rightarrow \text{Wife}(y,x)\) and a fact that \(A\) is the husband of \(B\), we can derive that \(B\) is the wife of \(A\). However, in many cases logic rules can be imperfect or even contradictory, and hence effectively modeling the uncertainty of logic rules is very critical. A more principled method for using logic rules is the Markov Logic Network (MLN) \cite{31,35}, which combines first-order logic and probabilistic graphical models. MLNs learn the weights of logic rules in a probabilistic framework and thus soundly handle the uncertainty. Such methods have been proven effective for reasoning on knowledge graphs. However, the inference process in MLNs is difficult and inefficient due to the complicated graph structure among triplets. Moreover, the results can be unsatisfactory as many missing triplets cannot be inferred by any rules.

Another type of methods for reasoning on knowledge graphs are the recent knowledge graph embedding based methods (e.g., TransE \cite{3}, DistMult \cite{44} and ComplEx \cite{40}). These methods learn useful embeddings of entities and relations by projecting existing triplets into low-dimensional spaces.

Preprint. Under review.
These embeddings preserve the semantic meanings of entities and relations, and can effectively predict the missing triplets. In addition, they can be efficiently trained with stochastic gradient descent. However, one limitation is that they do not leverage logic rules, which compactly encode domain knowledge and are useful in many applications.

We are seeking an approach that combines the advantages of both worlds, one which is able to exploit first-order logic rules while handling their uncertainty, infer missing triplets effectively, and can be trained in an efficient way. We propose such an approach called the probabilistic Logic Neural Networks (pLogicNet). A pLogicNet defines the joint distribution of a collection of triplets with a Markov Logic Network [31], which associates each logic rule with a weight and can be effectively trained with the variational EM algorithm [29]. In the variational E-step, we infer the plausibility of the unobserved triplets (i.e., hidden variables) with amortized mean-field inference [11, 18, 26], in which the variational distribution is parameterized as a knowledge graph embedding model. In the M-step, we update the weights of logic rules by optimizing the pseudolikelihood [1], which is defined on both the observed triplets and those inferred by the knowledge graph embedding model. The framework can be efficiently trained by stochastic gradient descent. Experiments on four benchmark knowledge graphs prove the effectiveness of pLogicNet over many competitive baselines.

2 Related Work

First-order logic rules can compactly encode domain knowledge and have been extensively explored for reasoning. Early methods such as expert systems [12, 15] use hard logic rules for reasoning. However, logic rules can be imperfect or even contradictory. Later studies try to model the uncertainty of logic rules by using Horn clauses [5, 17, 27, 42] or database query languages [28, 37]. A more principled method is the Markov logic network [35, 38], which combines first-order logic with probabilistic graphical models. Despite the effectiveness in a variety of tasks, inference in MLNs remains difficult and inefficient due to the complicated connections between triplets. Moreover, for predicting missing triplets on knowledge graphs, the performance can be limited as many triplets cannot be discovered by any rules. In contrast to them, pLogicNet uses knowledge graph embedding models for inference, which is much more effective by learning useful entity and relation embeddings.

Another category of approach for knowledge graph reasoning is the knowledge graph embedding method [3, 8, 25, 36, 40, 41, 44], which aims at learning effective embeddings of entities and relations. Generally, these methods design different scoring functions to model different relation patterns for reasoning. For example, the TransE method [3] defines each relation as a translation vector, which can effectively model the composition and inverse relation patterns. However, it cannot model the symmetric relations. The DistMult approach [44] models the symmetric relation with a bilinear scoring function. Nevertheless, it cannot deal with asymmetric relations. The ComplEx approach [40] models the asymmetric relations by using a bilinear scoring function in complex space. The recent RotatE approach [36] further models multiple relation patterns by defining each relation as a rotation in complex spaces. Overall, knowledge graph embedding methods are very effective and can be efficiently trained. However, they are not able to use logic rules, which are beneficial in many tasks. Recently, there are a few studies on combining logic rules and knowledge graph embedding [9, 14]. However, the uncertainty of logic rules is not modeled. Compared with them, pLogicNet is able to use logic rules and also handle their uncertainty in a principled way through Markov logic networks.

There are also some recent work using reinforcement learning for reasoning on knowledge graphs [6, 20, 34, 43], where an agent is trained to search for reasoning paths on knowledge graphs. However, training such agents is very hard, and most of these methods perform even worse than the knowledge graph embedding methods. Our pLogicNets are much easier to train and also more effective.

Lastly, there is also a recent study [29] on combining statistical relational learning and graph neural networks for semi-supervised node classification. Our work shares similar idea with this study, but we focus on a different problem, i.e., reasoning with first-order logic on knowledge graphs.

3 Preliminary

3.1 Problem Definition

A knowledge graph is a collection of relational facts, each of which is represented as a triplet \((h, r, t)\). Due to the high cost of knowledge graph construction, the coverage of knowledge graphs is usually limited. Therefore, a critical problem on knowledge graphs is to predict the missing facts.
Formally, given a knowledge graph \((E, R, O)\), where \(E\) is a set of entities, \(R\) is a set of relations, and \(O\) is a set of observed \((h, r, t)\) triplets, the goal is to infer the missing triplets by reasoning with the observed triplets. Following existing studies [24], the problem can be reformulated in a probabilistic way. Each triplet \((h, r, t)\) is associated with a binary indicator variable \(v_{(h,r,t)}\). \(v_{(h,r,t)} = 1\) means \((h, r, t)\) is true, and \(v_{(h,r,t)} = 0\) otherwise. Given some true facts \(v_O = \{v_{(h,r,t)} = 1\} | (h, r, t) \in O\), we aim to predict the labels of the remaining hidden triplets \(H\), i.e., \(v_H = \{v_{(h,r,t)}\} | (h, r, t) \in H\). We will discuss how to generate the hidden triplets \(H\) later in Sec. 4.4.

This problem has been extensively studied in both traditional logic rule-based methods and recent knowledge graph embedding methods. For logic rule-based methods, we mainly focus on one representative approach, the Markov logic network [31]. Essentially, both types of methods aim to model the joint distribution of the observed and hidden triplets \(p(v_O, v_H)\). Next, we briefly introduce the Markov logic network (MLN) [31] and the knowledge graph embedding methods [3, 41, 44].

### 3.2 Markov Logic Network

In the MLN, a Markov network is designed to define the joint distribution of the observed and the hidden triplets, where the potential function is defined by the first-order logic. Some common logic rules to encode domain knowledge include: (1) **Composition Rules.** A relation \(r_i\) is a composition of \(r_i\) and \(r_j\) means that for any three entities \(x, y, z\), if \(x\) has relation \(r_i\) with \(y\), and \(y\) has relation \(r_j\) with \(z\), then \(x\) has relation \(r_k\) with \(z\). Formally, we have \(\forall x, y, z \in E, v_{(x,r_i,y)} \land v_{(y,r_j,z)} \Rightarrow v_{(x,r_k,z)}\).

(2) **Inverse Rules.** A relation \(r_i\) is an inverse of \(r_j\) indicates that for two entities \(x, y\), if \(x\) has relation \(r_j\) with \(y\), then \(y\) has relation \(r_j\) with \(x\). We can represent the rule as \(\forall x, y \in E, v_{(x,r_j,y)} \Rightarrow v_{(y,r_j,x)}\).

(3) **Symmetric Rules.** A relation \(r\) is symmetric means that for any entity pair \(x, y\), if \(x\) has relation \(r\) with \(y\), then \(y\) also has relation \(r\) with \(x\). Formally, we have \(\forall x, y \in E, v_{(x,r,y)} \Rightarrow v_{(y,r,x)}\).

(4) **Subrelation Rules.** A relation \(r_j\) is a subrelation of \(r_i\) indicates that for any entity pair \(x, y\), if \(x\) and \(y\) have relation \(r_j\), then they also have relation \(r_i\). Formally, we have \(\forall x, y \in E, v_{(x,r_i,y)} \Rightarrow v_{(x,r_j,y)}\).

For each logic rule \(l\), we can obtain a set of possible groundings \(G_l\) by instantiating the entity placeholders in the logic rule with real entities in knowledge graphs. For example, for a subrelation rule, \(\forall x, y, v_{(x,Born in,y)} \Rightarrow v_{(x,Live in,y)}\), two groundings in \(G_l\) can be \(v_{(Newton,Born in,UK)} \Rightarrow v_{(Newton,Live in,UK)}\) and \(v_{(Einstein,Born in,German)} \Rightarrow v_{(Einstein,Live in,German)}\). We see that the former one is true while the latter one is false. To handle such uncertainty of logic rules, Markov logic networks introduce a weight \(w_l\) for each rule \(l\), and then the joint distribution of all triplets is defined as follows:

\[
p(v_O, v_H) = \frac{1}{Z} \exp \left( \sum_{l \in L} w_l \sum_{g \in G_l} \mathbb{1}\{g \text{ is true}\} \right) = \frac{1}{Z} \exp \left( \sum_{l \in L} w_l n_l(v_O, v_H) \right),
\]

where \(n_l\) is the number of true groundings of the logic rule \(l\) based on the values of \(v_O\) and \(v_H\).

With such a formulation, predicting the missing triplets essentially becomes inferring the posterior distribution \(p(v_H | v_O)\). Exact inference is usually infeasible due to the complicated graph structures, and hence approximate inference is often used such as MCMC [13] and loopy belief propagation [22].

### 3.3 Knowledge Graph Embedding

Different from the logic rule-based approaches, the knowledge graph embedding methods learn embeddings of entities and relations with the observed facts \(v_O\), and then predict the missing facts with the learned entity and relation embeddings. Formally, each entity \(e \in E\) and relation \(r \in R\) is associated with an embedding \(x_e\) and \(x_r\). Then the joint distribution of all the triplets is defined as:

\[
p(v_O, v_H) = \prod_{(h,r,t) \in O \cup H} \text{Ber}(v_{(h,r,t)} | f(x_h, x_r, x_t)),
\]

where \(\text{Ber}\) stands for the Bernoulli distribution, \(f(x_h, x_r, x_t)\) computes the probability that the triplet \((h, r, t)\) is true, with \(f(\cdot, \cdot, \cdot)\) being a scoring function on the entity and relation embeddings. For example in TransE, the \(f\) can be formulated as \(\sigma(\gamma - ||x_h + x_r - x_t||)\) according to [36], where \(\sigma\) is the sigmoid function and \(\gamma\) is a fixed bias. To learn the entity and relation embeddings, these methods typically treat observed triplets as positive examples and the hidden triplets as negative ones. In other words, they seek to maximize \(\log p(v_O = 1, v_H = 0)\). The whole framework can be efficiently optimized with the stochastic gradient descent algorithm.
where we employ a knowledge graph embedding model to infer the missing triplets, during which the exact inference is intractable, we approximate the true posterior distribution with a mean-field \[26\].

For inference, we aim to infer the posterior distribution of the hidden variables, i.e., 

\[ q(v_H) \]

as the learning procedure, we fix \[ q \] and update \[ p_w \] to maximize the evidence lower bound (ELBO) of the log-likelihood function, which is given as follows:

\[ \log p_w(v_O) \geq \mathcal{L}(q_\theta, p_w) = \mathbb{E}_{q_\theta(v_H)}[\log p_w(v_O, v_H) - \log q_\theta(v_H)], \]

where \[ q_\theta(v_H) \] is a variational distribution of the hidden variables \[ v_H \]. The equation holds when \[ q_\theta(v_H) \] equals to the true posterior distribution \[ p_w(v_H|v_O) \]. Such a lower bound can be effectively optimized with the variational EM algorithm \[23\], which consists of a variational E-step and an M-step. In the variational E-step, which is known as the inference procedure, we fix \[ p_w \] and update \[ q_\theta \] to minimize the KL divergence between \[ q_\theta(v_H) \] and \[ p_w(v_H|v_O) \]. In the M-step, which is known as the learning procedure, we fix \[ q_\theta \] and update \[ p_w \] to maximize the log-likelihood function of all the triplets, i.e., \[ \mathbb{E}_{q_\theta(v_H)}[\log p_w(v_O, v_H)] \]. Next, we introduce the details of both steps.

### 4.1 Variational EM

Given a set of first-order logic rules \[ L = \{ l_i \}_{i=1}^{|L|} \], our approach uses a Markov logic network \[51\] as in Eq. (1) to model the joint distribution of both the observed and hidden triplets:

\[ p_w(v_O, v_H) = \frac{1}{Z} \exp \left( \sum_l w_l n_l(v_O, v_H) \right), \]

where \[ w_l \] is the weight of rule \[ l \]. The model can be trained by maximizing the log-likelihood of the observed indicator variables, i.e., \[ \log p_w(v_O) \]. However, directly optimizing the objective is infeasible, as we need to integrate over all the hidden indicator variables \[ v_H \]. Therefore, we instead optimize the evidence lower bound (ELBO) of the log-likelihood function, which is given as follows:

\[ \log p_w(v_O) \geq \mathcal{L}(q_\theta, p_w) = \mathbb{E}_{q_\theta(v_H)}[\log p_w(v_O, v_H) - \log q_\theta(v_H)], \]

where \[ q_\theta(v_H) \] is a variational distribution of the hidden variables \[ v_H \]. The equation holds when \[ q_\theta(v_H) \] equals to the true posterior distribution \[ p_w(v_H|v_O) \]. Such a lower bound can be effectively optimized with the variational EM algorithm \[23\], which consists of a variational E-step and an M-step. In the variational E-step, which is known as the inference procedure, we fix \[ p_w \] and update \[ q_\theta \] to minimize the KL divergence between \[ q_\theta(v_H) \] and \[ p_w(v_H|v_O) \]. In the M-step, which is known as the learning procedure, we fix \[ q_\theta \] and update \[ p_w \] to maximize the log-likelihood function of all the triplets, i.e., \[ \mathbb{E}_{q_\theta(v_H)}[\log p_w(v_O, v_H)] \]. Next, we introduce the details of both steps.

### 4.2 E-step: Inference Procedure

For inference, we aim to infer the posterior distribution of the hidden variables, i.e., \[ p_w(v_H|v_O) \]. As exact inference is intractable, we approximate the true posterior distribution with a mean-field \[26\].
We can also use the observed triplets in $q_h(r,t)$, where $Ber$ stands for the Bernoulli distribution, and $f_p$ to optimize this objective, we first compute the log-likelihood function, i.e., $\log p(v_{h,r,t})$, the optimal $q_h(r,t)$ is given by the fixed-point condition as below (see appendix for proof):

$$\log q_h(r,t) = E_{q_h(r,t)}[\log p(v_{h,r,t})] + \text{const} \quad \text{for all } (h, r, t) \in H,$$  

where $MB(h, r, t)$ is the Markov blanket of $(h, r, t)$, which contains the triplets that appear together with $(h, r, t)$ in any grounding of the logic rules. For example, from a grounding $v_{\text{Newton,Live in,UK}} \Rightarrow v_{\text{Live in,UK}}$, we can know both triplets are in the Markov blanket of each other.

Based on the Eq. (5), our goal becomes finding a distribution $q_h(r,t)$ that satisfies the condition. However, Eq. (6) involves the expectation with respect to $q_h(r,t)$. To simplify the condition, we estimate the expectation by drawing a sample $\tilde{v}_{MB(h,r,t)} = \{\tilde{v}_{(h',r',t')} : (h',r',t') \in MB(h,r,t)\}$. Specifically, for each $(h', r', t') \in MB(h, r, t)$, if it is observed, we set $\tilde{v}_{(h',r',t')} = 1$, and otherwise $\tilde{v}_{(h',r',t')} \sim q_h(r,t)$. In this way, the right side of Eq. (6) is approximated as $\log p_w(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$, and thus the optimality condition can be further simplified as $q_h(r,t) \approx p_w(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$.

Intuitively, for each hidden triplet $(h, r, t)$, the knowledge graph embedding model predicts $v_{h,r,t}$ through the entity and relation embeddings (i.e., $q_h(r,t)$), while the logic rules make the prediction by utilizing the triplets connected with $(h, r, t)$ (i.e., $p_w(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$). If any triplet $(h', r', t')$ connected with $(h, r, t)$ is unobserved, we simply fill in $v_{(h',r',t')}$ with a sample $\tilde{v}_{(h',r',t')} \sim q_h(r,t)$. Then, the simplified optimality condition tells us that for the optimal knowledge graph embedding model, it should reach a consensus with the logic rules on the distribution of $v_{h,r,t}$ for every $(h, r, t)$, i.e., $q_h(r,t) \approx p_w(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$.

To learn the optimal $q_h$, we use a method similar to [32]. We start by computing $p_w(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$ with the current $q_h$. Then, we fix the value as target, and update $q_h$ to minimize the reverse KL divergence of $q_h(r,t)$ and the target $p(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$, leading to the following objective:

$$O_{U,L} = \sum_{(h,r,t) \in H} E_{p(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}}[\log q_h(r,t)].$$  

To optimize this objective, we first compute $p(v_{h,r,t}) \tilde{v}_{MB(h,r,t)}$ for each hidden triplet $(h, r, t)$. If the probability of $p(v_{h,r,t}) = 1$ is greater than 0.5, then we treat $(h, r, t)$ as a positive example and train the knowledge graph embedding model to maximize the log-likelihood $\log q_h(r,t) = 1$. Otherwise the triplet is treated as a negative example. In this way, the knowledge captured by logic rules can be effectively distilled into the knowledge graph embedding model.

We can also use the observed triplets in $O$ as positive examples to enhance the knowledge graph embedding model. Therefore, we also optimize the following objective function:

$$O_{U,L} = \sum_{(h,r,t) \in O} \log q_h(r,t) = 1.$$  

By adding Eq. 7 and 8, we obtain the overall objective function for $q_h$, i.e., $O_{\theta} = O_{U,L} + O_{U,L}$.

### 4.3 M-step: Learning Procedure

In the learning procedure, we will fix $q_h$, and update the weights of logic rules $w$ by maximizing the log-likelihood function, i.e., $E_{q_w(v(H))}[\log p_w(v(O \cup H\setminus(h,r,t)))]$. However, directly optimizing the log-likelihood function can be difficult, as we need to deal with the partition function, i.e., $Z$ in Eq. (5). Therefore, we follow existing studies [19, 31] and instead optimize the pseudolikelihood function [1]:

$$\ell_{PL}(w) \triangleq E_{q_w(v(H))}[\sum_{h,r,t} \log p_w(v_{h,r,t}) | v_{O \cup H\setminus(h,r,t)}]) = E_{q_w(v(H))}[\sum_{h,r,t} \log p_w(v_{h,r,t}) | v_{MB(h,r,t)}],$$
where the equation is derived from the independence property of the Markov logic network in Eq. 3.

We optimize \( w \) through the gradient descent algorithm. For each expected conditional distribution \( \mathbb{E}_{q^\theta(v)}[\log p_w(v) | \mathbf{v}_{\mathcal{M}B}(h,r,t)] \), suppose \( v(h,r,t) \) connects with \( \mathbf{v}_{\mathcal{M}B}(h,r,t) \) through a set of rules. For each of such rules \( l \), the derivative with respect to \( w_l \) is computed as (see appendix for proof):

\[
\nabla_{w_l} \mathbb{E}_{q^\theta(v)}[\log p_w(v) | \mathbf{v}_{\mathcal{M}B}(h,r,t)] \approx y(h,r,t) - p_w(v) = 1 | \mathbf{v}_{\mathcal{M}B}(h,r,t) \quad (9)
\]

where \( y(h,r,t) = 1 \) if \( (h, r, t) \) is an observed triplet and \( y(h,r,t) = q^\theta(v) = 1 \) if \( (h, r, t) \) is a hidden one. \( \mathbf{v}_{\mathcal{M}B}(h,r,t) = \{ \mathbf{v}(h',r',t') \} \) is a sample from \( q^\theta(v) \). For each \((h', r', t') \in \mathcal{M}B(h, r, t), \mathbf{v}(h', r', t') = 1 \) if \((h', r', t') \) is observed, and otherwise \( \mathbf{v}(h', r', t') \approx q^\theta(v(h', r', t')) \).

Intuitively, for each observed triplet \( (h, r, t) \in O \), we seek to maximize \( p_w(v(h,r,t) = 1 | \mathbf{v}_{\mathcal{M}B}(h,r,t)) \). For each hidden triplet \( (h, r, t) \in H \), we treat \( q^\theta(v(h,r,t) = 1) \) as target for updating \( p_w(v(h,r,t) = 1 | \mathbf{v}_{\mathcal{M}B}(h,r,t)) \). In this way, the knowledge graph embedding model \( q^\theta \) essentially provides extra supervision to benefit learning the weights of logic rules.

### 4.4 Optimization and Prediction

During training, we iteratively perform the E-step and the M-step until convergence. Note that there are a huge number of possible hidden triplets (i.e., \(|E| \times |R| \times |E| - |O|\)), and handling all of them is impractical for optimization. Therefore, we only include a small number of triplets in the hidden set \( H \). Specifically, an unobserved triplet \((h, r, t)\) is added to \( H \) if we can find a grounding \( \text{[premise]} \Rightarrow \text{[hypothesis]} \), where the hypothesis is \((h, r, t)\) and the premise only contains triplets in the observed set \( O \). In practice, we can construct \( H \) with brute-force search as in [14].

After training, according to the fixed-point condition given in Eq. 6, the posterior distribution \( p_w(v(h,r,t) | v_O) \) for \((h, r, t) \in H \) can be characterized by either \( q^\theta(v(h,r,t)) \) or \( p_w(v(h,r,t)) | \mathbf{v}_{\mathcal{M}B}(h,r,t) \) with \( \mathbf{v}_{\mathcal{M}B}(h,r,t) \sim q^\theta(v(h,r,t)) \). Although we try to encourage the consensus of \( p_w \) and \( q^\theta \) during training, they may still give different predictions as different information is used. Therefore, we use both of them for prediction, and we approximate the true posterior distribution \( p_w(v(h,r,t) | v_O) \) as:

\[
p_w(v(h,r,t) | v_O) \approx \frac{1}{2} \left( q^\theta(v(h,r,t)) + p_w(v(h,r,t) | \mathbf{v}_{\mathcal{M}B}(h,r,t)) \right) \quad (10)
\]

In practice, we also expect to infer the plausibility of the triplets outside \( H \). For each such triplet \((h, r, t)\), we can still compute \( q^\theta(v(h,r,t)) \) through the learned embeddings, but we cannot make predictions with the logic rules, so we simply replace \( p_w(v(h,r,t) = 1 | \mathbf{v}_{\mathcal{M}B}(h,r,t)) \) with 0.5 in Eq. 10.

### 5 Experiment

#### 5.1 Experiment Settings

**Datasets.** In experiments, we evaluate the pLogicNet on four benchmark datasets. The FB15k [5] and FB15k-237 [59] datasets are constructed from Freebase [2]. WN18 [3] and WN18RR [8] are constructed from WordNet [21]. The detailed statistics of the datasets are summarized in appendix.

**Evaluation Metrics.** We compare different methods on the task of knowledge graph reasoning. For each test triplet, we mask the head or the tail entity, and let each compared method predict the masked entity. Following existing studies [3, 44], we use the filtered setting during evaluation. The Mean Rank (MR), Mean Reciprocal Rank (MRR) and Hit@K (H@K) are treated as the evaluation metrics.

**Compared Algorithms.** We compare with both the knowledge graph embedding methods and rule-based methods. For the knowledge graph embedding methods, we choose five representative methods to compare with, including TransE [3], DistMult [44], HolE [25], ComplEx [40] and ConvE [8]. For the rule-based methods, we compare with the Markov logic network (MLN) [31] and the Bayesian Logic Programming (BLP) method [7], which model logic rules with Markov networks and Bayesian networks respectively. Besides, we also compare with RUGE [14] and NNE-AER [9], which are hybrid methods that combine knowledge graph embedding and logic rules. As only the results on the FB15k dataset are reported in the RUGE paper, we only compare with RUGE on that dataset. For our approach, we consider two variants, where pLogicNet uses only \( q^\theta \) to infer the plausibility of unobserved triplets during evaluation, while pLogicNet* uses both \( q^\theta \) and \( p_w \) through Eq. 10.
Experimental Setup of pLogicNet. To generate the candidate rules in the pLogicNet, we search for all the possible composition rules, inverse rules, symmetric rules and subrelations rules from the observed triplets, which is similar to \([10, 14]\). Then, we compute the empirical precision of each rule, i.e. \(p_l = \frac{|S_l \cap O|}{|S_l|}\), where \(S_l\) is the set of triplets extracted by the rule \(l\) and \(O\) is the set of the observed triplets. We only keep the rules whose empirical precision is more than 0.1. TransE \([3]\) is used as the knowledge graph embedding model to parameterize \(q_\theta\) by default, and we use the same hyperparameters as in \([36]\) during training. We update the weights of logic rules with gradient descent, and the learning rate is fixed as 0.0001.

5.2 Results

5.2.1 Comparing pLogicNet with Other Methods

Table 1: Results of reasoning on the FB15k and WN18 datasets. The results of the KGE and the Hybrid methods except for TransE are directly taken from the corresponding papers. H@K is in %.

<table>
<thead>
<tr>
<th>Category</th>
<th>Algorithm</th>
<th>FB15k</th>
<th>WN18</th>
</tr>
</thead>
<tbody>
<tr>
<td>KGE</td>
<td>TransE</td>
<td>40 0.730 64.5 79.3 86.4</td>
<td>272 0.772 70.1 80.8 92.0</td>
</tr>
<tr>
<td></td>
<td>DistMult</td>
<td>42 0.798 - - 89.3</td>
<td>655 0.797 - - 94.6</td>
</tr>
<tr>
<td></td>
<td>HoE</td>
<td>0.524 40.2 61.3 73.9</td>
<td>0.938 93.0 94.5 94.9</td>
</tr>
<tr>
<td></td>
<td>ComplEx</td>
<td>0.692 59.9 75.9 84.0</td>
<td>0.941 93.6 94.5 94.7</td>
</tr>
<tr>
<td></td>
<td>ConvE</td>
<td>0.657 55.8 72.3 83.1</td>
<td>0.943 93.5 94.6 95.6</td>
</tr>
<tr>
<td>Rule-based</td>
<td>BLP</td>
<td>415 0.242 13.1 26.9 42.4</td>
<td>706 0.643 73.7 71.7 83.0</td>
</tr>
<tr>
<td></td>
<td>MLN</td>
<td>515 0.321 21.0 37.0 55.0</td>
<td>- - - -</td>
</tr>
<tr>
<td>Hybrid</td>
<td>RUGE</td>
<td>0.768 70.3 81.5 86.5</td>
<td>- - - -</td>
</tr>
<tr>
<td></td>
<td>NNE-AER</td>
<td>0.803 76.1 83.1 87.4</td>
<td>0.943 94.0 94.5 94.8</td>
</tr>
<tr>
<td>Ours</td>
<td>pLogicNet</td>
<td>0.792 71.4 85.7 90.1</td>
<td>255 0.832 71.6 94.4 95.7</td>
</tr>
<tr>
<td></td>
<td>pLogicNet*</td>
<td>0.844 81.2 86.2 90.2</td>
<td>254 0.945 93.9 94.7 95.8</td>
</tr>
</tbody>
</table>

Table 2: Results of reasoning on the FB15k-237 and WN18RR datasets. The results of the KGE methods except for TransE are directly taken from the corresponding papers. H@K is in %.

<table>
<thead>
<tr>
<th>Category</th>
<th>Algorithm</th>
<th>FB15k-237</th>
<th>WN18RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>KGE</td>
<td>TransE</td>
<td>181 0.326 22.9 36.3 52.1</td>
<td>3410 0.223 1.3 40.1 53.1</td>
</tr>
<tr>
<td></td>
<td>DistMult</td>
<td>254 0.241 15.5 26.3 41.9</td>
<td>5110 0.43 39 44 49</td>
</tr>
<tr>
<td></td>
<td>ComplEx</td>
<td>339 0.247 15.8 27.5 42.8</td>
<td>5261 0.44 41 46 51</td>
</tr>
<tr>
<td></td>
<td>ConvE</td>
<td>244 0.325 23.7 35.6 50.1</td>
<td>4187 0.43 40 44 52</td>
</tr>
<tr>
<td>Rule-based</td>
<td>BLP</td>
<td>1985 0.092 6.2 9.8 13.0</td>
<td>12051 0.254 18.7 31.3 35.8</td>
</tr>
<tr>
<td></td>
<td>MLN</td>
<td>1980 0.098 6.7 10.3 16.0</td>
<td>11549 0.259 19.1 32.2 36.1</td>
</tr>
<tr>
<td>Ours</td>
<td>pLogicNet</td>
<td>173 0.330 23.1 36.9 52.8</td>
<td>3436 0.230 1.5 41.1 53.1</td>
</tr>
<tr>
<td></td>
<td>pLogicNet*</td>
<td>173 0.332 23.7 36.7 52.4</td>
<td>3408 0.441 39.8 44.6 53.7</td>
</tr>
</tbody>
</table>

The main results on the four datasets are presented in Tab. 1 and 2. We can see that the pLogicNet significantly outperforms the rule-based methods, as pLogicNet uses a knowledge graph embedding model to improve inference. pLogicNet also outperforms all the knowledge graph embedding methods in most cases, where the improvement comes from the capability of exploring the knowledge captured by the logic rules. Moreover, our approach is superior to both hybrid methods (RUGE and NNE-AER) under most metrics, as it can handle the uncertainty of logic rules in a principled way.

Comparing pLogicNet and pLogicNet*, pLogicNet* uses both \(q_\theta\) and \(p_w\) to predict the plausibility of hidden triplets, which outperforms pLogicNet in most cases. The reason is that the information captured by \(q_\theta\) and \(p_w\) is different and complementary, so combining them yields better performance.

5.2.2 Analysis of Different Rule Patterns

In pLogicNet, four types of rule patterns are used. Next, we systematically study the effect of each rule pattern. We take the FB15k and FB15k-237 datasets as examples, and report the results obtained with each single rule pattern in Tab. 3. On both datasets, most rule patterns can lead to significant improvement compared to the model without logic rules. Moreover, the effects of different rule patterns are quite different across datasets. On FB15k, the inverse and symmetric rules are more important, whereas on FB15k-237, the composition and subrelation rules are more effective.
Table 3: Analysis of different rule patterns. H@K is in %.

<table>
<thead>
<tr>
<th>Rule Pattern</th>
<th>FB15k</th>
<th>FB15k-237</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MR</td>
<td>MRR</td>
</tr>
<tr>
<td>Without</td>
<td>40</td>
<td>0.730</td>
</tr>
<tr>
<td>Composition</td>
<td>40</td>
<td>0.752</td>
</tr>
<tr>
<td>Inverse</td>
<td>39</td>
<td>0.813</td>
</tr>
<tr>
<td>Symmetric</td>
<td>40</td>
<td>0.793</td>
</tr>
<tr>
<td>Subrelation</td>
<td>40</td>
<td>0.761</td>
</tr>
</tbody>
</table>

5.2.3 Inference with Different Knowledge Graph Embedding Methods

Table 4: Comparison of using different knowledge graph embedding methods. H@K is in %.

<table>
<thead>
<tr>
<th>KGE Method</th>
<th>Algorithm</th>
<th>FB15k</th>
<th>FN18RR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MR</td>
<td>MRR</td>
<td>H@1</td>
</tr>
<tr>
<td>TransE</td>
<td>pLogicNet</td>
<td>33</td>
<td>0.792</td>
</tr>
<tr>
<td></td>
<td>pLogicNet</td>
<td>33</td>
<td>0.844</td>
</tr>
<tr>
<td>DistMult</td>
<td>pLogicNet</td>
<td>40</td>
<td>0.791</td>
</tr>
<tr>
<td></td>
<td>pLogicNet</td>
<td>39</td>
<td>0.815</td>
</tr>
<tr>
<td>ComplEx</td>
<td>pLogicNet</td>
<td>39</td>
<td>0.776</td>
</tr>
<tr>
<td></td>
<td>pLogicNet</td>
<td>45</td>
<td>0.788</td>
</tr>
</tbody>
</table>

In this part, we compare the performance of pLogicNet with different knowledge graph embedding methods for inference. We use TransE as the default model and compare with two other widely-used knowledge graph embedding methods, DistMult [44] and ComplEx [40]. The results on the FB15k and WN18RR datasets are presented in Tab. 4. Comparing with the results in Tab. 1 and 2, we see that pLogicNet improves the performance of all the three methods by using logic rules. Moreover, the pLogicNet achieves very robust performance with any of the three methods for inference.

Table 5: Effect of KGE on logic rules.

<table>
<thead>
<tr>
<th>Iteration</th>
<th>FB15k</th>
<th>WN18RR</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># Triplets</td>
<td>Precision</td>
</tr>
<tr>
<td>1</td>
<td>64,929</td>
<td>79.21%</td>
</tr>
<tr>
<td>2</td>
<td>74,717</td>
<td>79.31%</td>
</tr>
<tr>
<td>3</td>
<td>76,268</td>
<td>79.10%</td>
</tr>
</tbody>
</table>

5.2.4 Effect of Knowledge Graph Embedding on Logic Rules

In the M-step of pLogicNet, we use the learned embeddings to annotate the hidden triplets, and further update the weights of logic rules. Next, we analyze the effect of knowledge graph embeddings on logic rules. Recall that in the E-step, the logic rules are used to annotate the hidden triplets through Eq. (7), and thus collect extra positive training data for embedding learning. To evaluate the performance of logic rules, in each iteration we report the number of positive triplets discovered by logic rules, as well as the precision of the triplets in Tab. 5. We see that as training proceeds, the logic rules can find more triplets with stable precision. This observation proves that the knowledge graph embedding model can indeed provide effective supervision for learning the weights of logic rules.

5.2.5 Convergence Analysis

Finally, we present the convergence curves of pLogicNet on the FB15k and WN18 datasets in Fig. 2. The horizontal axis represents the iteration, and the vertical axis shows the value of Hit@1 (in %). We see that on both datasets, our approach takes only 2-3 iterations to converge, which is very efficient.

6 Conclusion

This paper studies knowledge graph reasoning, and an approach called the pLogicNet is proposed to integrate existing rule-based methods and knowledge graph embedding methods. pLogicNet models the distribution of all the possible triplets with a Markov logic network, which is efficiently optimized with the variational EM algorithm. In the E-step, a knowledge graph embedding model is used to infer the hidden triplets, whereas in the M-step, the weights of rules are updated based on the observed and inferred triplets. Experimental results prove the effectiveness of pLogicNet. In the future, we plan to explore more advanced models for inference, such as relational GCN [33] and RotatE [36].
References


