Neural Conditional Logic Field for Knowledge Bases

ABSTRACT

Combining logic reasoning and probabilistic inference for knowledge bases has been a long-standing goal for machine learning and AI. This task is made more challenging when we consider the presence of many unobserved facts, and complex entity types such as texts, audios and images. Existing approaches either lack of consistent probabilistic treatment of unobserved facts, difficult to scale to problems with a large number of entities, or are not able to handle complex entity types. In this paper, we propose a neural conditional logic field for consistent probabilistic logic inference, where the potentials of the graphical model are defined using neural tensorized logic formula (n-TLF), and the variational distribution is defined using a factor graph convolution neural networks (f-GCN). Probabilistic inference with f-GCN also allows us to design an efficient end-to-end stochastic training algorithm for the model parameters, which can scale up to millions of variables. Extensive experiments have demonstrated that our approach can significantly outperform existing approaches, and is much more scalable than previous state-of-the-art methods. In terms of the inference time on benchmark datasets, our method is 10 – 100 times faster than the competitor methods.

CCS CONCEPTS

• **Mathematics of computing → Markov networks**; **Variational methods**; • **Computing methodologies → Probabilistic reasoning**.

KEYWORDS

Logic reasoning; deep learning; probabilistic inference

ACM Reference Format:
. 2019. Neural Conditional Logic Field for Knowledge Bases. In *KDD '19: 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, August 04–08, 2019, Anchorage, Alaska, USA.* ACM, New York, NY, USA, 11 pages.

1 INTRODUCTION

Combining logic reasoning and probabilistic inference for knowledge bases has been a long-standing goal for machine learning and AI. Dominant paradigms considering these two aspects of AI typically use logic formulae to define graphical model components. For instance, probabilistic inductive logic programming [2] treats predicates as random variables, and uses a logic formula to define the conditional probability tables in directed graphical models. Markov Logic Networks [15] also define random variables in the same way, but uses a logic formula to define the potential functions in undirected graphical models. These seminal works have greatly extended the applicability of logic reasoning to handle partially correct logic formulae and noisy facts seen in real-world problems, while at the same time allowing probabilistic graphical models to exploit prior knowledge to learn in the region of small data.

However, these previous works are still limited in three aspects. First, to avoid the exponential cost of enumerating all possible combinations of unobserved facts, most existing approaches adopt the closed-world semantics when performing logic reasoning on reasonably large knowledge bases. The closed-world assumption simply assumes that all unobserved facts are negative facts, which clearly violates the reality of many knowledge bases where both positive and negative facts are explicitly collected, and one could provide a consistent probabilistic treatment of those unobserved facts using latent variables instead. Second, for knowledge bases with a large number of entities, it is challenging to carry out efficient inference. For instance, Markov Logic Networks need to first perform the *grounding* process, which explicitly instantiates all latent predicates before inference can be performed. The grounding process can be quadratic or even higher if the predicates involve a pair or more entities. Third, existing approaches rely on traditional logic techniques, such as SAT solver or MCMC sampling for inference, and are not able to handle entities in a mixture of discrete space (such as text) and continuous space (such as audios and images). As such knowledge bases are becoming increasingly common in modern applications, such as Visual Genome database with both knowledge graph and image patches [8], a unified probabilistic logic framework that can handle both entity types is much needed.

In this paper, we propose a neural conditional logic field (NCLF) to address the above challenges. Our method deals with open-world knowledge bases where unobserved facts are treated as latent variables in probabilistic graphical models. To deal with complex entity types, our method employs neural tensorized logic formula (n-TLF) as the potential functions for conditional random fields, and define a factor graph convolution neural networks (f-GCN) for performing variational inference. Probabilistic inference with f-GCN also allows us to design an efficient end-to-end stochastic training algorithm for the model parameters, which can scale up to millions of variables. We experiment on a number of benchmarks and large-scale synthetic datasets. Our approach outperforms significantly over existing approaches both in terms of the accuracy and efficiency under open-world semantics.

2 KNOWLEDGE BASES IN THE LANGUAGE OF LOGIC

In this section, we will express relational knowledge bases in the language of logic. This will facilitate our exposition of the neural conditional logic field later. Typically, a knowledge base $\mathcal{K}$ can be view as a tuple $\mathcal{K} = (\mathcal{C}, \mathcal{R}, \mathcal{O})$, consisting of a set $\mathcal{C}$ of entities, a set $\mathcal{R}$ of different relations, and a collection $\mathcal{O}$ of facts.

We will focus on generalized knowledge bases where complex entities and relations involving multiple entities are allowed. More specifically, in the language of first-order logic, our generalized knowledge bases consist of

• A set of M **constants** (entities), $\mathcal{C} = \{c_1, c_2, \ldots, c_M\}$, where each constant belongs to a finite or infinite space. For instance, a constant can be a discrete entity such as the identity of a person, a sample from a continuous space such as an image patch, or an object from a combinatorial space such as a drug molecular graph. Furthermore, multiple different types of constants can be simultaneously present in a single knowledge base.

• A set of N distinctive **predicates** (properties or relations), $\mathcal{R} = \{r_1, r_2, \ldots, r_N\}$, where each predicate is a logic function defined over the set $\mathcal{C}$ of constants, i.e.,

$$r(\cdot): \quad \underbrace{\mathcal{C} \times \mathcal{C} \times \ldots \times \mathcal{C}}_{\text{arity of the predicate, denoted as } |r|} \quad \mapsto [0,1]. \tag{1}$$

We note that the arity of a predicate can be thought of as the number of constants or arguments needed to instantiate a predicate. Thus the arity of a predicate can range from 1 all the way to the total number of constants, M. For instance,

– We may have a predicate $r(c) = \texttt{Person}(c)$ which checks whether an image patch constant c contains a person or not; or a predicate $r(c) = \texttt{Smoke}(c)$ which checks whether a person smoke or not. Typically, a singleton predicate like these examples return a logic variable indicating whether a constant has certain property or not.

– We may have a predicate $r(c, c') = \texttt{Friend}(c, c')$ which checks whether person c and c' are friend or not; or $r(c, c') = \texttt{OnTop}(c, c')$ which checks whether image patch c is on top of image patch c'. In general, predicates can be asymmetric in terms of their arguments, i.e., different positions of the constants have different semantic meaning. For instance, for the $\texttt{OnTop}(\cdot)$ predicate, exchanging the position of the two constants, c, c', will result in very different meaning, and hence different logic value.

– We may have higher order predicates that contain more than two constants such as $r(c, c', c'') = \texttt{Transaction}(c, c', c'')$ which checks whether company c sells c' product c''.

When a particular set of constants are assigned to the arguments of a predicate, the predicate is called a **grounded predicate**. One can think of

each grounded predicate ≡ a random variable,

which will be used to define our probabilistic model later. We will denote an assignment of constants to the arguments of a predicate as a_r. For instance, for a grounded predicate $r(c, c')$ with constant c and c', $a_r = (c, c')$ and we can simply write $r(a_r) := r(c, c')$.

Since each argument of a predicate r can be assigned a constant from $\mathcal{C}$, there are many ways one can instantiate or ground a predicate. For instance, for a predicate $r(\cdot, \cdot)$, there are $M \times M$ way to ground it by assigning each constant from $\mathcal{C}$ to each argument of r. In general, for a d-ary predicate, there are M^d ways to assign constants to the predicate. We will denote the entire collection of assignments of constants to a predicate r as $\mathcal{A}_r = \{a_r^1, a_r^2, \ldots\}$.

• A set of L **observed facts**, $\mathcal{O} = \{o_1, \ldots, o_L\}$, where each observation is associated with a probability in the interval $[0,1]$. This probability reflects the confidence that the fact is true. In this way, our generalized knowledge bases have consistent probabilistic treatment. For observed fact without uncertainty, which is simply positive or negative fact, the probability just goes to binary and can be either 1 or 0. For instance,

– We may have a label o indicating image patch c is not a person, i.e., $o \equiv [\texttt{Person(c) = 0}]$;

– We may have the information o to verify that person c and c' are indeed friends, i.e., $o \equiv [\texttt{Friend(c, c') = 1}]$;

– We may have a record o that company c never traded product c'' with company c', i.e., $o \equiv [\texttt{Transaction}(c, c', c'') = 0]$.

Since there are many grounded predicates, the set of observed facts are typically much smaller than those **unobserved facts**. We adopt the **open-world** paradigm, where we will treat these

unobserved facts ≡ latent variables

in knowledge bases.

• A set of K first-order **logic formulae**, $\mathcal{F} = \{f_1, \ldots f_K\}$, where each formula is a logic function defined over the set $\mathcal{C}$ of constants, i.e.,

$$f(\cdot): \quad \underbrace{\mathcal{C} \times \mathcal{C} \times \ldots \times \mathcal{C}}_{\text{arity of the formula, denoted as } |f|} \quad \mapsto [0,1]. \tag{2}$$

We note that the arity of a formula is the number of constants or arguments needed to evaluate a logic formula. Typically, a logic formula is defined via the composition of a few predicates. For instance, a logic formula f can be

$$f(c, c') = \texttt{Smoke}(c) \wedge \texttt{Friend}(c, c') \Rightarrow \texttt{Smoke}(c'), \tag{3}$$

whose arity is 2. Equivalently, based on the De Morgan's law, this formula can be transformed into a disjunctive form

$$\neg\texttt{Smoke}(c) \vee \neg\texttt{Friend}(c, c') \vee \texttt{Smoke}(c'),$$

where $\neg$ denotes the negation of a predicate. In general, let $\mathcal{R}_f^+$ and $\mathcal{R}_f^-$ be the set of predicates and predicate negations in a logic formula respectively, then f can be written as

$$f = \left(\bigvee\nolimits_{r \in \mathcal{R}_f^+} r\right) \bigvee \left(\bigvee\nolimits_{r \in \mathcal{R}_f^-} \neg r\right), \tag{4}$$

where we denote $\mathcal{R}_f = \mathcal{R}_f^+ \cup \mathcal{R}_f^-$ as the complete set of predicates used in the logic formula f. Typically, the sizes of the two sets $\mathcal{R}_f^+$ and $\mathcal{R}_f^-$ are not large, since each logic formula only involves a small number of predicates.

To avoid notational clumsiness, we formulate the logic formula in disjunctive form as above. This can be easily extended to general conjunctive normal form (CNF), which is a conjunction of disjunctions. It is known that any first-order logic formula can be converted into CNF using a mechanical sequence of steps [16]. In Section 4.1, we will further explain how our framework can be extended to support arbitrary logic formula in CNF. We assume that all the logic formulae are universally quantified. For existentially quantified logic formulae, we can handle them by reducing each formula to a disjunction of all of its possible instantitations.

Similar to the case of a predicate, there are many ways that one can assign constants to the arguments of a logic formula. We will denote an assignment of constants to a formula f as a_f, and the entire collection of consistent assignments of constants as $\mathcal{A}_f = \{a_f^1, a_f^2, \ldots\}$.

Despite the elegance of the above formalism of knowledge bases using logic representations, such logical formalism does not fully expose the structures present in the observed facts and logic formulae. For instance, it is not easy to express logic variable binding constraints, e.g., to require the grounding of one logic variable being different from another logic variable. Also, it is not straightforward to specify the logic variables to be shared among multiple predicates. Thus, in the next section, we will present a factor graph view of the observed facts and logic formulae, which will provide the basis for our factor graph convolution neural networks (f-GCN) and neural tensorized logic formula (n-TLF) potentials.

3 KNOWLEDGE BASES AS FACTOR GRAPHS

As an alternative but more clear representation, we will use factor graph to express the collection of observed facts and the set of logic formulae in a knowledge base. More specifically, factor graph representation for the observed facts will be a bipartite graph $\mathcal{G}_{\mathcal{K}} = (\mathcal{C}, \mathcal{O}, \mathcal{E})$ where nodes on one side of the graph will correspond to the set of constant $\mathcal{C}$ while nodes on the other side of the graph will correspond to the set of observed facts $\mathcal{O}$, which is called *factor* in this case. The set of T edges, $\mathcal{E} = \{e_1, \ldots, e_T\}$, will connect constants and the observed facts. More specifically, an edge

> $e = (c, o, i)$ between node c and o exists, if the grounded predicate associated with o uses c as an argument in its i-th argument position.

An illustration of a toy factor graph can be found in Figure 1. The upper nodes are fact nodes, and the bottom nodes are constant (entity) nodes. As we discussed in Section 2, each fact is associated with a probability in the interval $[0, 1]$. The interfaces on fact nodes are associated with the argument position in the corresponding predicate. The edges have encoded the information of argument positions of logic variables, i.e., we assign different edge types for different argument position. In Figure 1, we use darker (respectively, brighter) color for edges that connect to the first (respectively, second) argument position.

A logic formula f can also be represented as a factor graph, $\mathcal{G}_f = (\mathcal{C}_f, \mathcal{R}_f, \mathcal{E}_f)$, where nodes on one side of the graph will correspond to a set of distinct constants $\mathcal{C}_f$ needed in the formula, while nodes on the other side will correspond to the set of predicates $\mathcal{R}_f$ used to define the formula. The set of T_f edges, $\mathcal{E}_f$, will connect constants to predicates or predicate negation. That is, an edge

> $e = (c, r, i)$ between node c and predicate r exists, if the predicate r use constant c in its i-th argument.

We note that the set of distinctive constants used in the definition of logic formula are templates where actual constant can be instantiated from $\mathcal{C}$.

An illustration of logic formula factor graph can be found in Figure 2. In this illustration, we have a single logic formula

$$\texttt{Husband(X,Y)} \wedge \texttt{Mother(Y,Z)} \Rightarrow \texttt{Daughter(Z,X)}.$$

Using the De Morgan’s law, this formula can be transformed into disjunctive form:

$$\neg\texttt{Husband(X,Y)} \vee \neg\texttt{Mother(Y,Z)} \vee \texttt{Daughter(Z,X)}.$$

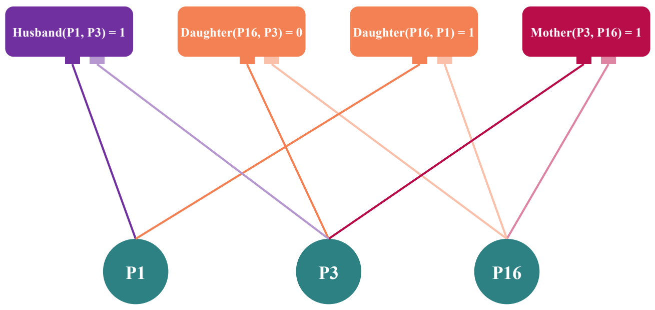

Figure 1: Illustration of a toy factor graph. The upper nodes are fact nodes, and the bottom nodes are constant (entity) nodes.

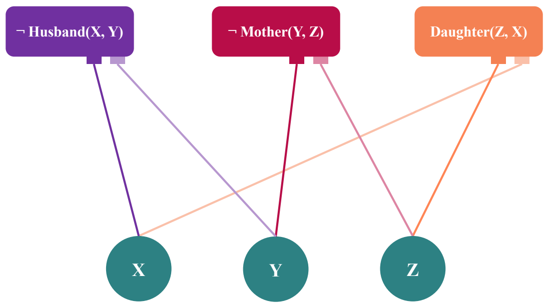

Figure 2: Illustration of a single-formula factor graph.

Similar to the factor graph for observed facts, we also differentiate the type of edges by the position of argument.

4 NEURAL CONDITIONAL LOGIC FIELD

In this section, we will explain our neural conditional logic field. It defines a joint conditional distribution over the entire collection of grounded predicates in a knowledge base (a world). More specifically, given the set of constants, $\mathcal{C}$, the set of predicates, $\mathcal{R}$, and their associated sets of allowable assignments of constants, $\mathcal{A}_r$ for each $r \in \mathcal{R}$, the entire set of variables, $\mathcal{X}$, involved in the conditional distribution is

$$\mathcal{X} := \{r(a_r) \mid r \in \mathcal{R}, a_r \in \mathcal{A}_r\}\,. \tag{5}$$

That is for each predicate $r \in \mathcal{R}$, we instantiate r with each possible assignment a_r of constants consistent with the definition of r, and each of these instantiated predicates will correspond to a binary random variable in the neural conditional logic field.

Furthermore, due to our adoption of the open-world semantics, some variables in $\mathcal{X}$ are observed but most of them will be unobserved. Essentially, the set of observed variables will correspond to those observed facts $\mathcal{O}$, meaning that these variables have been assigned a definite truth value. Then the set of unobserved variables is $\mathcal{H} := \mathcal{X} \setminus \mathcal{O}$.

We note that since each formula is defined via a set of predicates (see Eq. (4) and the example in Eq. (3)), a grounded formula will then involve a set of grounded predicates (or a set of random variables from Eq. (5)). More specifically, if a formula f is defined via the set of predicates $\mathcal{R}_f$, then the set of variables involved in a grounded formula is $\cup_{r \in \mathcal{R}_f}\{r(a_r)\}$. Since each predicate variable $r(a_r)$ may be involved in multiple formulae with other variables, these variables are coupled in a complex way via the formulae.

Now we can define the conditional distribution of random variables in $\mathcal{X}$ given the set of constants C as

$$P(\mathcal{X} \mid C) = \frac{1}{Z(C)} \exp\left(\sum_{f \in \mathcal{F}} w_f \sum_{a_f \in \mathcal{A}_f} \phi_f(a_f)\right) \tag{6}$$

$$= \frac{1}{Z(C)} \exp\Big(\Psi\big(\{r(a_r)\}\big)\Big), \tag{7}$$

where $\phi_f(\cdot)$ is a neural tensorized logic function defined later, and $Z(C)$ is a normalization constant

$$Z(C) = \sum_{r \in \mathcal{R}} \sum_{a_r \in \mathcal{A}_r} \sum_{r(a_r) \in [0,1]} \exp\Big(\Psi\big(\{r(a_r)\}\big)\Big)$$

summing over all grounded predicates, and all possible truth values assigned to each predicate. We note that we have made the distribution $P(\mathcal{X}|C)$ and the normalization factor $Z(C)$ depend on the constant features. This is especially useful when the constants involve complex data types such as images and texts. As we will explain later, both the neural potential function $\phi_f(\cdot)$ and the inference network can take complex data types into account. Roughly, a learnable embedding function will be applied to these complex constant types before we use their embeddings to define the potential functions. Furthermore, in our joint distribution definition, variables corresponding to both observed and unobserved facts are taken into account, allowing consistent probabilistic inference to be carried out for both.

Next, we provide details on the definition of the potential function $\phi_f(\cdot)$, the variational inference framework, and the factor graph convolution neural networks essential to the inference.

4.1 Nerual Tensorsized Logic Formula

We will first define a neural tensor model for predicates, and then further define the neural tensor model for logic formulae. More specifically, we will parameterize each constant c as an embedding vector $h = h(c) \in \mathbb{R}^d$ where $h(\cdot)$ is a neural network which takes a constant as input, and output a vector representation of the constant in $\mathbb{R}^d$. For instance, if the constant is an image patch, then the embedding function $h(\cdot)$ can be a convolutional neural network (CNN) extracting features from image patches [9, 10]. If the constant is a scientific paper, then the embedding function $h(\cdot)$ can be a Long Short Term Memory (LSTM) network extracting features from natural language texts [6, 23].

With the embedding of the constants, we can define the feature function of a singleton predicate $r(c)$ as follows

$$\phi_r(c) = \begin{cases} \sigma(\psi_r(c)) \cdot r(c), & \text{if } r(c) > 0.5, \\ 1 - \sigma(\psi_r(c)) \cdot (1 - r(c)), & \text{otherwise,} \end{cases} \tag{8}$$

where $\sigma(\cdot)$ is a sigmoid function, $\psi_r(c) = u_r^\top \tanh(V_r h + b_r)$ with $h = h(c)$ being the embedding of constant c, $V_r \in \mathbb{R}^{d \times d}$ and $b_r, u_r \in \mathbb{R}^d$ being matrix and vector parameters respectively. Here we can interpret $\sigma(\cdot)$ as a parameterized scoring function that learns to assign confidence score for each ground predicate. To ensure the symmetry of confidence score for any predicate and its negation (with a flipped truth value), we branch the feature function based on whether the predicate takes a truth value over 0.5, as shown above in Eq. (8).

Similarly, we can define the feature function for a predicate $r(c, c')$ involving two constants as follows

$$\phi_r(c, c') = \begin{cases} \sigma(\psi_r(c, c')) \cdot r(c, c'), & \text{if } r(c, c') > 0.5, \\ 1 - \sigma(\psi_r(c, c')) \cdot (1 - r(c, c')), & \text{otherwise,} \end{cases} \tag{9}$$

where $\psi_r(c, c') = u_r^\top \tanh(h^\top \mathcal{W}_r h' + V_r(h + h') + b_r)$ with $h = h(c)$ and $h' = h(c')$ being the embedding of constant c and c' respectively, $\mathcal{W}_r$ being a tensor parameters with size $\mathbb{R}^{d \times d \times d}$, $V_r \in \mathbb{R}^{d \times d}$ and $b_r, u_r \in \mathbb{R}^d$ being matrix and vector parameters respectively.

Similarly, using higher order tensor with a concatenation of the embeddings of multiple constants, we can also define neural potentials for higher order predicates as follows

$$\phi_r(a_r) = \begin{cases} \sigma(\psi_r(a_r)) \cdot r(a_r), & \text{if } r(a_r) > 0.5, \\ 1 - \sigma(\psi_r(a_r)) \cdot (1 - r(a_r)), & \text{otherwise,} \end{cases} \tag{10}$$

where we denote the constant assignment as $a_r = (c, c', c'', \ldots)$, and $\psi_r(a_r) = u_r^\top \tanh(\mathbf{h}^\top \mathcal{W}_r \mathbf{h} + V_r \mathbf{h} + b_r)$ with $\mathbf{h} = \mathbf{h}(a_r) = [h(c); h(c'); h(c''); \ldots] \in \mathbb{R}^{|r| \cdot d}$ being the column concatenation of the embedding vector of each constant in a_r, $\mathcal{W}_r$ being a tensor parameters with size $\mathbb{R}^{|r| \cdot d \times |r| \cdot d \times d}$, $V_r \in \mathbb{R}^{d \times |r| \cdot d}$ and $b_r, u_r \in \mathbb{R}^d$ being matrix and vector parameters respectively.

With neuralized potentials for predicates, we now define the neuralized potentials for formulae. Take the following formula in disjunctive form as an example

$$f(c, c') = \neg\texttt{Smoke}(c) \vee \neg\texttt{Friend}(c, c') \vee \texttt{Smoke}(c').$$

We employ soft logic operator [1, 27] to define its potential as

$$\begin{aligned} \phi_f(c, c') = \max\{\, & 1 - \phi_{\texttt{Smoke}}(c), \\ & 1 - \phi_{\texttt{Friend}}(c, c'), \\ & \phi_{\texttt{Smoke}}(c') \,\}, \end{aligned} \tag{11}$$

where the disjunction has been replaced by the max $\{\cdot\}$ operator, which is a commonly used s-norm (also known as t-co-norm) operator. Furthermore, each predicates r and its negation in the logic formula has been replaced by the corresponding neural potential function ϕ_r and $1 - \phi_r$ respectively.

In general, if we have a formula which is expressed in disjunctive form in Eq. (4), the corresponding neuralized potential can be defined as

$$\phi_f = \max\left\{ \max_{r \in \mathcal{R}_f^+} \{\phi_r\}, \max_{r \in \mathcal{R}_f^-} \{1 - \phi_r\} \right\}. \tag{12}$$

As we discussed before, the formula can be extended to a more general form, i.e., conjunctive normal form (CNF), so that any first-order logic formula can be handled. To make this happen, we just need to employ a t-norm operator such as min $\{\cdot\}$ to approximate the logical conjunction, and compute the potential of CNF formula using both the s-norm and t-norm operators.

Next we describe a variational inference framework for learning the parameters in these neural tensorized logic formula.

4.2 Variational Inference Framework

Since we have both observed and unobserved variables in our neural conditional logic field, we will use EM algorithm to learn the parameters in the model. More specifically, the variational evidence lower bound (ELBO) for those observed predicates in O can be written as (for simplicity of notation, we will omit the conditioning on C from now on)

$$\begin{aligned}\log P(O) \geqslant\ & \mathbb{E}_{Q(\mathcal{H}\mid O)}[\log P(O,\mathcal{H})] \\ & - \mathbb{E}_{Q(\mathcal{H}\mid O)}[\log Q(\mathcal{H}\mid O)], \end{aligned} \tag{13}$$

where $Q(\mathcal{H}|O)$ is the posterior distribution of the latent variables given the observed variables. We note that the computation of the posterior distribution $Q(\mathcal{H}|O)$ is a challenging inference problem, since it involves grounding of all predicates, and then perform graphical model inference in the grounded graph with many variables. We address the challenge by employing two techniques: one is the traditional mean field variational inference which approximate the posterior distribution by a product distribution, and the other is a factor graph convolutional neural network which we will explain in more details in the next section.

In the mean field inference framework, we will approximate the joint distribution of latent variables as a product distribution. That is

$$Q(\mathcal{H}|O) \approx \prod_{r(a_r)\in\mathcal{H}} Q(r(a_r)|O). \tag{14}$$

With this variational distribution:

- The first term in the ELBO in Eq. (13) becomes

$$\begin{aligned}& \mathbb{E}_{Q(\mathcal{H}|O)}[\log P(O,\mathcal{H})] \\ =& \sum_{f\in\mathcal{F}}\sum_{a_f\in\mathcal{A}_f}\mathbb{E}_{\prod_{r\in\mathcal{R}_f}Q(r(a_r)|O)}[\phi_f(a_f)]. \end{aligned} \tag{15}$$

The above simplified expression has the form of summation over formulae, and over all possible assignment of constants to each formula. Thus, this double summation may involve a large number of terms. Furthermore, for each term, we will need to compute the posterior distribution $Q(r(a_r)|O)$. We will address the large number of terms using the stochastic training procedure in Section 5, and the efficient posterior inference for $Q(r(a_r)|O)$ using the factor graph convolution neural network in the next section.

- And the second term in the ELBO in Eq. (13) becomes

$$\begin{aligned}& -\mathbb{E}_{Q(\mathcal{H}\mid O)}[\log Q(\mathcal{H}\mid O)] \\ =& -\sum_{r(a_r)\in\mathcal{H}}\mathbb{E}_{Q(r(a_r)|O)}[\log Q(r(a_r)|O)], \end{aligned} \tag{16}$$

which is the sum of entropies of the variational posterior distributions $Q(r(a_r)|O)$. This also involves a large number of terms since the summation needs to range over all possible unobserved facts. Typically, the number of latent facts are much larger than those observed facts. Similarly, we will address the large number of terms using the stochastic training procedure in Section 5, and the efficient posterior inference for $Q(r(a_r)|O)$ using the factor graph convolution neural network in the next section.

4.3 Factor Graph Convolution Neural Networks

Previous probabilistic logic inference techniques, such as lifted inference for Markov logic networks [22], make use of local graph similarity or graph symmetry, to reduce posterior inference computation and define new inference graph. However, lifted inference is still computationally intensive since the lifting process requires all predicates be grounded in the first place. Inspired by the lifted inference which exploits local graph similarly, and to void the full grounding process, we will instead define the factor graph convolution neural networks (f-GCN) for efficient variational posterior inference. The goal of the f-GCN is to capture node similarity based on the local graph around that node. Recent research shows that graph convolution neural networks can learn to perform graph isomorphism check and the learned features have the ability to distinguish graphs of different topology [24], suggesting f-GCN could lead to an effective and efficient approximation for lifted inference.

Our f-GCN network will compute a p-dimensional embedding μ_c for each constant node c and an embedding μ_o for each fact node o in the factor graph $\mathcal{G}_{\mathcal{K}} = (C, O, \mathcal{E})$. Our f-GCN defines the network architecture recursively according to the factor graph structure $\mathcal{G}_{\mathcal{K}}$. After a few steps of recursion, the network will produce an embedding for each node, taking into account both graph characteristics and long-range interactions between these nodes. More specifically, our f-GCN will update the embedding in the following fashion

$$\mu_c^t \leftarrow \gamma\left(W_1\,\mu_c^{t-1} + W_2\sum_{o\in\mathcal{N}(c)}\mu_o^{t-1}\right), \forall c \tag{17}$$

$$\mu_o^t \leftarrow \gamma\left(W_3\,\mu_o^{t-1} + W_4\sum_{c\in\mathcal{N}(o)}\mu_c^{t-1}\right), \forall o \tag{18}$$

where W_1, W_2, W_3, W_4 are parameters to be learned, and $\mathcal{N}(\cdot)$ is the set of neighbors of a node in the factor graph $\mathcal{G}_{\mathcal{K}}$, and $\gamma(\cdot)$ is a generic nonlinear activation such as a rectified linear unit.

Based on the embedding updating formula, one can see that the embedding update process is carried out based on the factor graph topology. A new round of embedding sweeping across the nodes will start only after the embedding update for all nodes from the previous round has finished. It is easy to see that the update also defines a process where the node embeddings are propagated to other nodes via the nonlinear propagation steps in Eq. (17) and Eq. (18). Furthermore, the more update iterations one carries out, the farther away the node embeddings will propagate and get aggregated nonlinearly at distant nodes. In the end, if one terminates after T iterations, each node embedding μ^T will contain information about its T-hop neighborhood as determined by the factor graph topology. An illustration of one iteration of graph embedding can be found in Figure 3. Node embedding can be initialized with raw features of facts and constants. For example, as illustrated in Figure 3, each fact node is associated with an embedding of the corresponding predicate and the truth value of the fact. Similarly, each constant node is associated with its feature vector, which can be one-hot vector for discretized symbols, or CNN extracted features for image patches, or pre-trained word embeddings for word objects, etc. All these initial node embeddings can be mapped to a unified embedding space, so that different types of constant nodes can be handled together in our framework.

After we obtain the embedding μ_c^T for each constant c, we can use these embeddings to define the varitional posterior distribution of the hidden variables, $Q(r(a_r)|O)$. More specifically, suppose $r(a_r)$ is

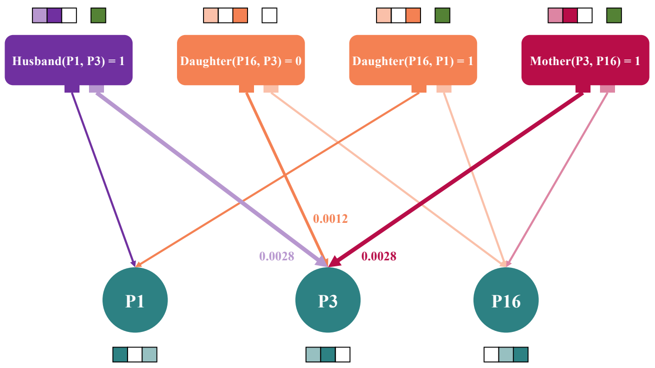

Figure 3: Illustration of one graph convolution iteration on the factor graph. Gradients are obtained from the Kinship experiment. The directed edges show the direction of gradients, and the thickness of edge indicates the magnitude of gradients. Both the upper fact nodes and the bottom constant nodes are associated with feature vectors as the initial node embeddings.

a d-ary predicate involving constant assignment $a_r = (c, c', c'', \dots)$, then we can define the variational posterior as

$$\begin{aligned} &Q(r(a_r) = 1|O) \\ =&\sigma\left(\left[\mu_c^T; \mu_{c'}^T; \mu_{c''}^T; \dots\right]^\top W_5 \left[\mu_c^T; \mu_{c'}^T; \mu_{c''}^T; \dots\right] + b_5\right), \end{aligned} \tag{19}$$

where $\sigma(\cdot)$ is the sigmoid unit, $\left[\mu_c^T; \mu_{c'}^T; \mu_{c''}^T; \dots\right]$ is a column concatenation of the node embedding vectors, W_5 and b_5 are parameters to be learned.

The advantages of defining the variational posterior using factor graph convolution neural network are twofold. First, the f-GCN is performed over the graph defined by a small number of observed facts, typically linear in the number of constants. This sparse graph enables f-GCN to run efficiently and capture graph features. Second, the posterior distribution over any predicate can be easily produced by making use of the embeddings of those involved constants. Especially, in the case when we want to sample just a small set of posterior distributions, we can just produce these sampled posteriors without the need to compute the posteriors for all possible grounded predicates. This ability to produce posterior distributions on request also forms the basis for our later efficient training algorithms.

5 EFFICIENT LEARNING

We will learn the parameters in the neural tensorized logic formula (n-TLF) and the factor graph convolution neural networks (f-GCN) jointly by maximizing the variational ELBO. However, the two terms in ELBO, Eq. (15) and Eq. (16), still contain summations over many terms: $\sum_{\mathcal{A}_f}$ requires to sum over all instantiation of each formula, and $\sum_{r(a_r)\in\mathcal{H}}$ requires to sum over all latent variables. One can overcome this challenge by approximating the ELBO with stochastic sampling.

First, maximizing the ELBO term in Eq. (15) is equivalent to maximize

$$\frac{1}{|\mathcal{F}|}\sum_{f\in\mathcal{F}} w_f \frac{1}{|\mathcal{A}_f|}\left(\sum_{a_f\in\mathcal{A}_f}\mathbb{E}\left[\phi_f(a_f)\right]\right), \tag{20}$$

where $|\mathcal{F}|$ and $|\mathcal{A}_f|$ are the total number of formulae and the total number of all groundings of each formula. And this can be approximated by a sampled batch of ground formulae as

$$\frac{1}{N_b}\sum_{f\in\mathcal{F}_b} w_f \left(\sum_{a_f\in\mathcal{A}_{b,f}}\mathbb{E}\left[\phi_f(a_f)\right]\right), \tag{21}$$

where N_b is the number of ground formulae in the batch b, and $\mathcal{F}_b$, $\mathcal{A}_{b,f}$ are the batch formula space and substitution space respectively.

Second, sampling for the second term in ELBO in Eq. (16), can be coupled with the sampling of ground formula: as one sampled a batch of ground formulae, the set of latent variables to be summed over in Eq. (16) is just the set of latent variables appearing in the ground formula batch. Thus Eq. (16) can be approximated by

$$\sum_{r(a_r)\in\mathcal{H}_b}\mathbb{E}\left[\log Q(r(a_r)|O)\right], \tag{22}$$

where $\mathcal{H}_b$ is the set of latent variables in batch b. Coupling these two samples are sensible, as this sampling strategy is essentially equivalent to sample a sub-graph from the complete factor graph, and approximate the complete graph ELBO with that of the sub-graph.

This sub-graph sampling technique essentially breaks down the exponential summations by approximating it with a sequence of summations with a controllable number of terms, making it highly efficient for training the neuralized factor graph and formula using a standard differentiable stochastic variational inference setting: sample a minibatch of data, compute the ELBO, do stochastic gradient descent and repeat until it converges.

One remaining difficulty is that the efficient sampling over the space of all ground formulae can be still nontrivial. As there are exponential many formulae, one cannot sample by explicitly enumerating all the formulae and permute them. On the other hand, not all ground formulae will contribute to the optimization during training. For example, a ground formula with only observed variables will not accumulate gradient to the f-GCN, as evaluating this formula is independent of the latent variable posterior. For efficient training, we propose the following sampling scheme: 1) uniformly samples a formula f from the space of $\mathcal{F}$; 2) shuffle the space of its predicate space $\mathcal{R}_f$; 3) for each predicate r popped from the top of the $\mathcal{R}_f$, with a probability of p_{obs} we instantiate it as an observed variable and with a probability of $1-p_{obs}$ it will become a uniformly sampled variable; 4) if r is observed, then we uniformly samples a fact associate this predicate if any can be found in the knowledge base, otherwise, or if r is not observed, we simply substitute its logic variables with random constants. When a formula is fully grounded, we examine its form and reject those without latent variables. In practice, the p_{obs} is usually set to a large value such as 0.9. The intuition is that one wants to prioritize on sampling formulae containing both observed and latent variables. Otherwise, in the cases where a formula is fully latent, f-GCN is essentially optimizing towards learning the prior distribution determined by the form of formula and its weight, which is unlikely to be close to the actual posterior distribution.

6 EXPERIMENTS

In this section, we demonstrate NCLF's ability in deductive inference task over two real-world datasets with open-world semantics. NCLF is evaluated by both area under the precision-recall curve (AUC-PR) score and wallclock time, and is compared against five strong baselines. Next we analyze its scalability by evaluating the inference time over a synthetic dataset of different sizes. Finally, we conduct a study to investigate the benefit of n-TLF against the classical discrete logic.

Benchmark Datasets. We evaluate NCLF on large datasets chosen from benchmarks that are commonly used in MLN inference: 1) the social network dataset UW-CSE [15], which contains publicly available information of students and professors in the CSE department of UW. The dataset is split into five sets according to the home department (AI, Graphics, Language, System and Theory) of the entities. The goal is to predict who is whose advisor from information about class taught, authors of publications, etc. UW-CSE provides 94 hand-coded first-order logic formulae Note that these crowd-sourced formulae are not as clean as being always true, but are typically true; 2) the entity resolution dataset Cora [19], which consists of a collection of citations to computer science research papers. The dataset is also split into five subsets according to the field of research. The task is to de-duplicate citations, author fields, and venue fields referring to the same underlying entity. Cora also provides 46 hand-coded first-order logic rules, such as if two fields have high TF-IDF similarity, they are probably the same.

Additionally, to analyze the scalability of NCLF, we also introduce a synthetic dataset that resembles the popular Kinship dataset [3]. The original dataset contains kinship relationships (e.g., `Father`, `Brother`) among 24 family members in the Alyawarra tribe from Central Australia. Given the kinship relationships, the model is tasked to infer the gender of all entities, which makes the task non-trivial. The synthetic Kinship dataset preserves all the logic predicates. We hand-coded 22 first-order logic formulae (listed in Table 6) and make the number of entities to be controllable. For example, to generate a dataset with n entities, we randomly split n entities into two groups which represent the first and second generation respectively. Within each group, entities are grouped into a few sub-groups representing the sister- and brother-hood. Finally, entities from different sub-groups in the first generation are randomly coupled and a sub-group in the second generation is assigned to them as their children. To generate the knowledge base, one traverse this family tree, and record all kinship relations for each entity. In this experiment, we generate five datasets by linearly increasing the number of entities. Detailed statistics of the benchmark and synthetic datasets are provided in Table 1.

Competitor Methods. We evaluate our proposed method against a number of state-of-the-art inference algorithms for Markov Logic Networks:

- MCMC (Gibbs Sampling): the most widely used method for approximate inference in the field of Markov networks [5], which is also popular in the Markov Logic Networks literature [15].
- Belief Propagation (BP): another well-known and widely used method for approximate probabilistic inference of graphical models [26].
- Lifted Belief Propagation (Lifted BP): a lifted version of belief propagation algorithm, which constructs a lifted network to greatly reduce the cost of inference [22].
- MC-SAT: an efficient MCMC algorithm that combines slice sampling with satisfiability testing, which significantly outperforms standard Gibbs sampling and simulated tempering methods [13].
- Hinge-Loss Markov Random Field (HL-MRF): a recently proposed probabilistic graphical model that generalizes the MRF inference to convex objective, which has been demonstrated to be much more scalable than traditional inference methods [1].

Experimental Settings. All experiments are conducted on a GPU-enabled (Nvidia RTX 2080 Ti) Linux machine powered by Intel Xeon Silver 4116 processors at 2.10GHz with 256GB RAM. For all experiments, we use 10% queries for validation, and we report results on the test set. For NCLF, we implement it using Pytorch and train it with Adam optimizer [7]. The learning rate is set to 0.0005 with sample batch size 8. Dimensionality of all node and edge embeddings in f-GCN is set to 128. Output of n-TLF is clipped towards 0 or 1 and the weights are fixed. Steps of propagation in f-GCN are set to 1. Increasing the number of propagation steps may slightly improve the performance but slow down the inference speed. Note that for the three benchmarks that we experiment on, many queries can be inferred by just considering the factor nodes within one-step neighborhood. For example, in the UW-CSE dataset we have $\texttt{Student}(s) = 0$ as a negative observed fact, and the query $\texttt{AdvisedBy}(p, s)$ can be inferred false (or low probability) according to logic formula $\texttt{AdvisedBy(X,Y)} \Rightarrow \texttt{Student(Y)}$. In the corresponding factor graph, the constant node s and the fact node $\texttt{Student}(s)$ are directly connected, and the one-step propagation in f-GCN can extract sufficient information for the inference.

Evaluation metrics. We adopt the area under the precision-recall curve (AUC-PR) as the evaluation metric for inference accuracy. Due to the severe imbalance of positive and negative samples in typical logic reasoning tasks, the AUC-PR is better than the AUC-ROC to reflect the actual model performance, and is widely used in the literature [15]. We use wallclock running time in minutes to evaluate the inference efficiency. For all experiments, we limit the running time to 24 hours.

6.1 Inference with Open-World Semantics

We evaluate NCLF and the 5 baselines on the deductive inference task over UW-CSE and Cora datasets under open-world semantics.

Inference accuracy. Table 2 summarizes the AUC-PR scores on benchmark and synthetic datasets. A hyphen in the entry indicates that the inference is either out of memory or exceeds the time limit (24 hours). Note that since the lifted BP is guaranteed to get identical results as the BP algorithm [22], the results of these two methods are merged into one row. For UW-CSE, the results suggest that NCLF consistently outperforms all 5 baselines. For Cora, We report the average AUC-PR score over the 5 splits of Cora dataset for NCLF, as none of the baselines methods is feasible under open-world setting[1]. On synthetic Kinship, since the dataset is noise-free and formulae hold true for all observed facts, HL-MRF method achieves the score of 1 for the first 4 datasets. NCLF yields similar but not perfect

[1]We report the performance of all baselines under closed-world setting in Table 5 as a sanity check, which follows the same setting as in their original works.

Table 1: Statistics of the benchmark and synthetic datasets. The statistics of Cora is averaged over its five splits.

Counting	**Kinship**					**UW-CSE**					**Cora**
	S1	S2	S3	S4	S5	AI	Graphics	Language	Systems	Theory	(average)
# entity	62	110	160	221	266	300	195	82	277	174	616
# relation	15	15	15	15	15	22	22	22	22	22	10
# fact	187	307	482	723	885	731	449	182	733	465	12K
# query	38	62	102	150	183	4K	4K	1K	5K	2K	2K
# ground atom	50K	158K	333K	635K	920K	95K	70K	15K	95K	51K	157K
# ground formula	550K	3M	9M	23M	39M	73M	64M	9M	121M	54M	457M

Table 2: Inference performance on three benchmark datasets.

Method	**Kinship**					**UW-CSE**					**Cora**
	S1	S2	S3	S4	S5	AI	Graphics	Language	Systems	Theory	(average)
MCMC	0.5263	-	-	-	-	-	-	-	-	-	-
BP / Lifted BP	0.5263	0.5806	0.5490	0.5467	0.5628	0.0076	0.0054	0.0115	0.0064	0.0067	-
MC-SAT	0.5405	0.6000	0.5490	0.5467	-	0.0301	0.0519	0.0569	0.0146	0.0168	-
HL-MRF	**1.0000**	**1.0000**	**1.0000**	**1.0000**	-	0.0615	0.0611	0.0216	0.0411	0.0250	-
NCLF	**1.0000**	0.9965	0.9942	0.9992	**0.9974**	**0.0665**	**0.1543**	**0.3242**	**0.0572**	**0.1056**	**0.6020**

scores for all but the first set, presumably caused by the stochastic nature of our sampling and optimization method.

Inference efficiency. The inference time on the UW-CSE dataset is summarized in Table 3, while the time of Cora is omitted as none of the baseline is feasible. Over all five splits of the dataset, NCLF outperforms the all baseline methods by at least one or two orders of magnitude.

Table 3: Inference time on the UW-CSE dataset.

Method	**Inference Time** (minutes)				
	AI	Graphics	Language	Systems	Theory
MCMC	>24h	>24h	>24h	>24h	>24h
BP	408	352	37	457	190
Lifted BP	321	270	32	525	243
MC-SAT	172	147	14	196	86
HL-MRF	135	132	18	178	72
NCLF	**5**	**3**	**2**	**2**	**3**

6.2 Model Scalability

The inference time of NCLF and 5 baseline methods over five synthetic datasets are illustrated in Figure 4. As the size of the dataset grows linearly, inference time of all 5 baseline methods grows exponentially. Slower methods such as MCMC and BP becomes infeasible for large datasets. And for HL-MRF, while maintaining a short wallclock time, it exhibits an exponential increase in the space complexity. For dataset S5, it ran out of memory with our testing platform. On the other hand, NCLF maintains a nearly constant

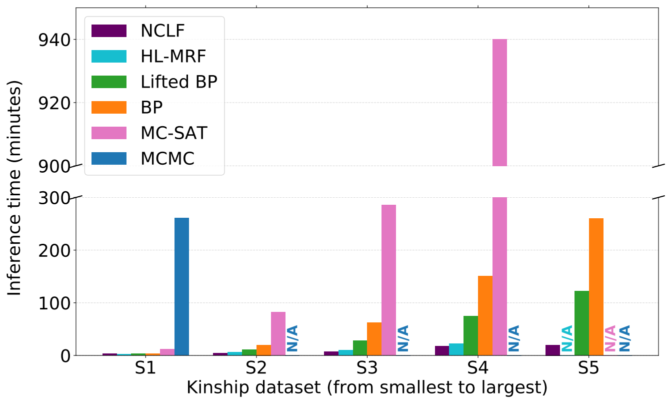

Figure 4: Inference time on the Kinship datasets. N/A indicates that the inference is either running out of memory or exceeds the time limit.

inference time with the increasing size of the dataset, demonstrating strong scalability.

6.3 Reasoning with n-TLF

To demonstrate the benefit of n-TLF over the classical discrete logic, we evaluate the NCLF over the 5 splits of the Cora dataset under two settings: 1) NCLF, we clip the output of Eq. (8) and Eq. (10) to be strictly 0 or 1, which is equivalent to the potential function in standard MLN network; and 2) NCLF-soft, we jointly learn the $\psi_r(\cdot)$ function and the posterior model in an end-to-end fashion. The AUC-PR scores of the two settings are shown in Table 4. NCLF-soft outperforms the discrete version in each split, and leads to a 9.5% increase on average AUC-PR score.

6.4 Visual Genome?

6.5

Table 4: Performance of our method on the Cora dataset.

Method	Cora					
	S1	S2	S3	S4	S5	(average)
NCLF	0.6182	0.7080	0.4497	0.5762	0.6578	0.6020
NCLF-soft	**0.6906**	**0.7366**	**0.4958**	**0.6358**	**0.7380**	**0.6594**

7 RELATED WORK

Markov Logic Networks: The seminal paper of Markov Logic Networks [15] have bridged the gap between logic reasoning and probabilistic inference. Since then, MLNs have been applied to a variety of applications, including entity resolution [20], social networks [28], information extraction [14], etc. Many works appear in the literature to improve the original MLN in both inference accuracy [12, 19] and inference efficiency [1, 13, 21, 22]. Nevertheless, to date, MLNs still struggle to handle large-scale knowledge bases in practice, mainly because of the exponential cost of constructing ground Markov networks and the NP-hard inference problem.

Neural symbolic reasoning: A lot of recent works propose different approaches to combine deep learning with symbolic logic reasoning. TensorLog [28] formulates logic reasoning as a series of matrix multiplication, and is able to learn chain-like logic rules with an attention-based neural controller [25]. Logic Tensor Networks [17, 18] build a fully differentiable neural network regularized by first-order logic rules, which can be applied to knowledge completion task. DeepProbLog [11] proposes a probabilistic logic programming language that incorporates deep learning with neural predicates. ∂ILP [4] uses end-to-end deep learning model to perform inductive logic programming with pre-defined rule templates.

8 CONCLUSIONS AND FUTURE WORK

In this work, we propose a framework named neural conditional logic field (NCLF) for probabilistic logic inference. We define the potentials of the graphical model using neural tensorized logic formula (n-TLF), and define the variational distribution using a factor graph convolution neural networks (f-GCN). With our fully differentiable formulation, our framework can be trained end-to-end in stochastic training fashion. Our method can scale up to millions of variables, which is highly efficient and scalable compared to existing methods. In the future, we plan to further explore our framework in mainly two directions: 1) experiment on high-dimensional constants in continuous space, such as images and audios, and handle a mixture of different entity types; 2) explore how to perform structure learning or rule induction using our framework, so that we can learn new rules from existing knowledge bases, and this may help discover interesting new facts and again help rule induction as a virtuous learning cycle.

REFERENCES

[1] Stephen H Bach, Matthias Broecheler, Bert Huang, and Lise Getoor. 2015. Hinge-loss markov random fields and probabilistic soft logic. *arXiv preprint arXiv:1505.04406* (2015).

[2] Luc De Raedt and Kristian Kersting. 2008. Probabilistic inductive logic programming. In *Probabilistic Inductive Logic Programming*. Springer, 1–27.

[3] Woodrow W Denham. 1973. *The detection of patterns in Alyawara nonverbal behavior*. Ph.D. Dissertation. University of Washington, Seattle.

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

[5] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. 1995. *Markov chain Monte Carlo in practice*. Chapman and Hall/CRC.

[6] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] 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. 2016. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 1097–1105.

[10] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[11] Robin Manhaeve, Sebastijan Dumančić, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. 2018. DeepProbLog: Neural Probabilistic Logic Programming. *arXiv preprint arXiv:1805.10872* (2018).

[12] Lilyana Mihalkova and Raymond J Mooney. 2007. Bottom-up learning of Markov logic network structure. In *Proceedings of the 24th international conference on Machine learning*. ACM, 625–632.

[13] Hoifung Poon and Pedro Domingos. 2006. Sound and efficient inference with probabilistic and deterministic dependencies. In *AAAI*, Vol. 6. 458–463.

[14] Hoifung Poon and Pedro Domingos. 2007. Joint inference in information extraction. In *AAAI*, Vol. 7. 913–918.

[15] Matthew Richardson and Pedro Domingos. 2006. Markov logic networks. *Machine learning* 62, 1-2 (2006), 107–136.

[16] Stuart J Russell and Peter Norvig. 2016. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,.

[17] Luciano Serafini, Ivan Donadello, and Artur d'Avila Garcez. 2017. Learning and reasoning in logic tensor networks: theory and application to semantic image interpretation. In *Proceedings of the Symposium on Applied Computing*. ACM, 125–130.

[18] Luciano Serafini and Artur d'Avila Garcez. 2016. Logic tensor networks: Deep learning and logical reasoning from data and knowledge. *arXiv preprint arXiv:1606.04422* (2016).

[19] Parag Singla and Pedro Domingos. 2005. Discriminative training of Markov logic networks. In *AAAI*, Vol. 5. 868–873.

[20] Parag Singla and Pedro Domingos. 2006. Entity resolution with markov logic. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 572–582.

[21] Parag Singla and Pedro Domingos. 2006. Memory-efficient inference in relational domains. In *AAAI*, Vol. 6. 488–493.

[22] Parag Singla and Pedro M Domingos. 2008. Lifted First-Order Belief Propagation.. In *AAAI*, Vol. 8. 1094–1099.

[23] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

[24] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2018. How Powerful are Graph Neural Networks? *arXiv preprint arXiv:1810.00826* (2018).

[25] Fan Yang, Zhilin Yang, and William W Cohen. 2017. Differentiable learning of logical rules for knowledge base completion. *CoRR, abs/1702.08367* (2017).

[26] Jonathan S Yedidia, William T Freeman, and Yair Weiss. 2001. Generalized belief propagation. In *Advances in neural information processing systems*. 689–695.

[27] Lotfi A Zadeh. 1975. Fuzzy logic and approximate reasoning. *Synthese* 30, 3-4 (1975), 407–428.

[28] Weizhe Zhang, Xiaoqiang Li, Hui He, and Xing Wang. 2014. Identifying network public opinion leaders based on markov logic networks. *The scientific world journal* 2014 (2014).

A INFERENCE WITH CLOSED-WORLD SEMANTICS FOR BASELINE METHODS

We report the inference results with closed-world setting on UW-CSE and Cora dataset for all 5 baseline methods. In other words, the predicates observed in the knowledge base is assumed to be closed, meaning for all instantiations of these predicates that do not appear in the knowledge base are considered negative. And only the query predicate remains open-world.

Since this setting is generally adopted by the original works of these baselines, we conduct these experiments and compare the results with that reported in the original works for sanity checking. The results are summarized in Table 5. And we found the results are similar to those reported in the original works. One can find the AUC-PR scores compared to those (Table 2) under open-world setting are actually better. This is due to the way the datasets were originally collected and evaluated generally complies with the closed-world assumption. In our future work, we plan to evaluate NCLF and baselines with more challenging and inherently open-world datasets.

B LOGIC FORMULAE IN KINSHIP DATASET

A full list of the first-order logic formulae in the Kinship dataset is summarized in Table 6.

Table 5: Inference performance of MCMC, (Lifted) BP, MC-SAT and HL-MRF methods under the closed-world semantics.

Method	Cora					UW-CSE				
	S1	S2	S3	S4	S5	AI	Graphics	Language	Systems	Theory
MCMC	0.4282	0.6273	0.2419	0.4563	0.5628	0.1912	0.0386	0.0286	0.1506	0.0777
BP / Lifted BP	0.4351	0.6228	0.2430	0.4473	0.5650	0.2096	0.0413	0.0108	0.1353	0.0504
MC-SAT	0.4271	0.6287	0.2429	0.4572	0.5683	0.1348	0.0356	0.0289	0.1129	0.0814
HL-MRF	0.5971	0.7765	0.5234	0.7021	0.8097	0.2562	0.1847	0.0618	0.2719	0.1907

Table 6: Full list of first-order logic formulae in the Kinship dataset.

First-Order Logic Formula
Father(X,Z) ∧ Mother(Y,Z) ⇒ Husband(X,Y)
Father(X,Z) ∧ Husband(X,Y) ⇒ Mother(Y,Z)
Husband(X,Y) ⇒ Wife(Y,X)
Son(Y,X) ⇒ Father(X,Y) ∨ Mother(X,Y)
Daughter(Y,X) ⇒ Father(X,Y) ∨ Mother(X,Y)
Father(X,Y) ⇒ Son(Y,X) ∨ Daughter(Y,X)
Mother(X,Y) ⇒ Son(Y,X) ∨ Daughter(Y,X)
Son(Y,X) ⇒ Child(Y,X)
Daughter(Y,X) ⇒ Child(Y,X)
Husband(X,Y) ⇒ Male(X)
Husband(X,Y) ⇒ Female(Y)
Wife(X,Y) ⇒ Female(X)
Wife(X,Y) ⇒ Male(Y)
Father(X,Y) ⇒ Male(X)
Mother(X,Y) ⇒ Female(X)
Son(X,Y) ⇒ Male(X)
Daughter(X,Y) ⇒ Female(X)
Female(X) ∧ Child(Y,X) ⇒ Mother(X,Y)
Male(X) ∧ Child(Y,X) ⇒ Father(X,Y)
Female(X) ∧ Child(X,Y) ⇒ Daughter(X,Y)
Male(X) ∧ Child(X,Y) ⇒ Son(X,Y)
Male(X) ⇒ ¬Female(X)
Female(X) ⇒ ¬Male(X)