RNNLogic: Learning Logic Rules for Reasoning on Knowledge Graphs

Anonymous Authors[1]

Abstract

This paper studies learning logic rules for reasoning on knowledge graphs. Logic rules provide interpretable explanations when used for prediction as well as being able to generalize to other tasks, and hence are critical to learn. Existing methods either suffer from the problem of searching in a large search space (e.g., neural logic programming) or ineffective optimization due to sparse rewards (e.g., techniques based on reinforcement learning). To address these limitations, in this paper we propose a principled probabilistic model called RNNLogic. RNNLogic treats logic rules as latent variables, and simultaneously trains a rule generator as well as a reasoning predictor with logic rules. Specifically, an EM algorithm is developed for optimization. In the E-step, a set of high-quality rules are selected by the rule generator and reasoning predictor via posterior inference, which help reduce the search space significantly; in the M-step, both the rule generator and reasoning predictor are updated with the selected high-quality rules in the E-step. Extensive experiments on four benchmark datasets prove the effectiveness of RNNLogic.

1. Introduction

Knowledge graphs are collections of real-world facts, which are useful in various applications. Each fact is typically specified as a triplet $(h, \texttt{r}, t)$ or equivalently $\texttt{r}(h, t)$, meaning entity h has relation $\texttt{r}$ with entity t. For example, *Bill Gates* is the `Co-founder` of *Microsoft*. As it is impossible to collect all facts, knowledge graphs are incomplete. Therefore, a fundamental problem on knowledge graphs is to predict missing facts by reasoning with existing ones, a.k.a. knowledge graph reasoning.

This paper focuses on learning logic rules for reasoning on knowledge graphs. For example, one may extract a rule $\forall X, Y, Z \ \ \texttt{hobby}(X, Y) \leftarrow \texttt{friend}(X, Z) \wedge \texttt{hobby}(Z, Y)$, meaning that if Z is a friend of X and Z has hobby Y, then Y is also likely the hobby of X. Afterwards, the rule can be applied to infer new hobbies of different people. Such logic rules are able to improve interpretability as well as precision of the reasoning process (Qu & Tang, 2019; Zhang et al., 2020). Moreover, logic rules can also be reused and generalized to other domains and datasets (Teru & Hamilton, 2020). However, due to the large search space of logic rules, inferring high-quality logic rules for reasoning on knowledge graphs is a challenging task.

Indeed, a variety of methods have been proposed for learning logic rules from knowledge graphs. For example, the path ranking algorithm (Lao & Cohen, 2010; Lao et al., 2011) enumerates relational paths between entities as candidate logic rules, and further learns a weight for each rule as an assessment of rule qualities. There are also some recent methods based on neural logic programming (Sadeghian et al., 2019; Yang et al., 2017; Yang & Song, 2020), which are able to learn logic rules and their weights at the same time in a differentiable way. Though empirically effective for prediction, the search space of these methods is exponentially large, making it difficult to identify high-quality logic rules. On the other hand, some recent efforts (Das et al., 2018; Lin et al., 2018; Shen et al., 2018; Xiong et al., 2017) formulate the problem as a sequential decision making process, and use reinforcement learning to search for logic rules on knowledge graphs, which significantly reduces search complexity. However, due to the large action space and sparse reward in training, the performance of these methods is not yet satisfying.

In this paper, we propose a principled probabilistic approach called RNNLogic which overcomes the above limitations. Our approach consists of a rule generator as well as a reasoning predictor with logic rules, which are simultaneously trained to enhance each other. The rule generator provides high-quality logic rules that are used for training the reasoning predictor, while the reasoning predictor provides effective reward to train the rule generator, which helps significantly reduce the search space. Specifically, for each query-answer pair, e.g., $\mathbf{q} = (h, r, ?)$ and $\mathbf{a} = t$, we model the probability of the answer conditioned on the query and existing knowledge graph $\mathcal{G}$, i.e., $p(\mathbf{a}|\mathcal{G}, \mathbf{q})$, where a set of

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

logic rules $\mathbf{z}$ is treated as a latent variable. The rule generator defines a prior distribution over logic rules for each query, i.e., $p(\mathbf{z}|\mathbf{q})$, which is parameterized by a recurrent neural network (Hochreiter & Schmidhuber, 1997). The reasoning predictor computes the likelihood of the answer conditioned on the logic rules and the existing knowledge graph $\mathcal{G}$, i.e., $p(\mathbf{a}|\mathcal{G}, \mathbf{q}, \mathbf{z})$. We develop an EM algorithm (Koller & Friedman, 2009; Neal & Hinton, 1998) for training. In the E-step, a set of high-quality logic rules are selected according to the posterior distribution. In the M-step, the rule generator is updated to imitate the high-quality rules selected in the E-step, and the reasoning predictor is updated with the selected logic rules. Experimental results on four knowledge graph benchmarks show that RNNLogic outperforms existing state-of-the-art methods for reasoning on knowledge graphs. Besides, RNNLogic is also able to generate high-quality logic rules.

2. Model

In this section, we introduce the proposed approach RNNLogic which learns logic rules for knowledge graph reasoning. We first formally define knowledge graph reasoning and logic rules.

Knowledge Graph Reasoning. Let $p_{\text{data}}(\mathcal{G}, \mathbf{q}, \mathbf{a})$ denote the training data distribution, where $\mathcal{G}$ is an existing knowledge graph characterized by a set of $(h, \mathtt{r}, t)$-triplets, which we may also write as $\mathtt{r}(h, t)$, $\mathbf{q} = (h, \mathtt{r}, ?)$ is a query, and $\mathbf{a} = t$ is a target answer. Given $\mathcal{G}$ and the query $\mathbf{q}$, the goal is to predict the correct answer $\mathbf{a}$. More formally, we aim to model the probabilistic distribution $p(\mathbf{a}|\mathcal{G}, \mathbf{q})$.

Logic Rule. We perform knowledge graph reasoning by learning logic rules, where logic rules in this paper have the conjunctive form $\forall \{X_i\}_{i=0}^{l}\ \mathtt{r}(X_0, X_l) \leftarrow \mathtt{r}_1(X_0, X_1) \wedge \cdots \wedge \mathtt{r}_l(X_{l-1}, X_l)$ with l being the rule length. This syntactic structure naturally captures composition, and can easily express other common logic rules such as symmetric or inverse rules. For example, let $\mathtt{r}^{-1}$ denote the inverse relation of relation $\mathtt{r}$, then each symmetric rule can be expressed as $\forall \{X, Y\}\ \mathtt{r}(X, Y) \leftarrow \mathtt{r}^{-1}(X, Y)$.

In RNNLogic, we treat a set of logic rules which could explain a query as a latent variable we have to infer. To do this, we introduce a rule generator and a reasoning predictor using logic rules. Given a query, the rule generator employs a recurrent neural network to generate a set of logic rules, which are given to the reasoning predictor for prediction. We optimize RNNLogic with an EM algorithm. In the E-step, a few important logic rules are identified via posterior inference, with the prior defined by the rule generator and likelihood specified by the reasoning predictor. In the M-step, the rule generator and the reasoning predictor are both updated with the high-quality rules selected in the E-step.

2.1. Probabilistic Formalization

We start by formalizing knowledge graph reasoning in a probabilistic way, where a *set of logic rules* $\mathbf{z}$ is treated as a latent variable. The target distribution $p(\mathbf{a}|\mathcal{G}, \mathbf{q})$ is jointly modeled by a rule generator and a reasoning predictor. The rule generator p_θ defines a prior over a set of latent rules $\mathbf{z}$ conditioned on a query $\mathbf{q}$, while the reasoning predictor p_w gives the likelihood of the answer $\mathbf{a}$ conditioned on latent rules $\mathbf{z}$, the query $\mathbf{q}$, and the knowledge graph $\mathcal{G}$. Then $p(\mathbf{a}|\mathcal{G}, \mathbf{q})$ can be naturally computed as:

$$p_{w,\theta}(\mathbf{a}|\mathcal{G}, \mathbf{q}) = \sum_{\mathbf{z}} p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \mathbf{z}) p_\theta(\mathbf{z}|\mathbf{q}). \tag{1}$$

The goal is to jointly train the rule generator and reasoning predictor to maximize the likelihood of training data. Formally, the objective function is presented as below:

$$\max_{\theta, w} \mathcal{O}(\theta, w) = \mathbb{E}_{(\mathcal{G}, \mathbf{q}, \mathbf{a}) \sim p_{\text{data}}}[\log p_{w,\theta}(\mathbf{a}|\mathcal{G}, \mathbf{q})]. \tag{2}$$

2.2. Parameterization

Rule Generator. The rule generator defines the distribution $p_\theta(\mathbf{z}|\mathbf{q})$. For a query $\mathbf{q}$, the rule generator aims at generating a set of latent logic rules $\mathbf{z}$ for answering the query.

To make the rules more general, given a query $\mathbf{q} = (h, \mathtt{r}, ?)$, we generate the set of compositional logic rules by only considering the query relation $\mathtt{r}$ without the query entity h. For each compositional rule in the abbreviation form $\mathtt{r} \leftarrow \mathtt{r}_1 \wedge \cdots \wedge \mathtt{r}_l$, it can naturally be viewed a sequence of different relations $[\mathtt{r}, \mathtt{r}_1, \mathtt{r}_2 \cdots \mathtt{r}_l, \mathtt{r}_{\text{END}}]$, where $\mathtt{r}$ is the query relation or the head of the rule, $\{\mathtt{r}_i\}_{i=1}^{l}$ are the body of the rule, and $\mathtt{r}_{\text{END}}$ is a special relation indicating the end of the relation sequence.

Such relation sequences can be effectively modeled by recurrent neural networks (Hochreiter & Schmidhuber, 1997), and hence we introduce RNN_θ to parameterize the rule generator. Given a query relation $\mathtt{r}$, RNN_θ will sequentially generate each relation in the body of a rule, until it reaches the ending relation $\mathtt{r}_{\text{END}}$. In this process, the probability of the generated rule can be naturally computed. Based on such probabilities of rules, we can further define the distribution over a set of rules $\mathbf{z}$ with the following multinomial distribution:

$$p_\theta(\mathbf{z}|\mathbf{q}) = \text{Mu}(\mathbf{z}|N, \text{RNN}_\theta(\cdot|\mathtt{r})), \tag{3}$$

where Mu stands for multinomial distributions, N is a hyperparameter for the size of the rule set $\mathbf{z}$, and $\text{RNN}_\theta(\cdot|\mathtt{r})$ defines the probability distributions over all the compositional rules with rule head being $\mathtt{r}$. The generative process of a rule set $\mathbf{z}$ is quite intuitive. Basically, we sample from the distribution over rules defined by $\text{RNN}_\theta(\cdot|\mathtt{r})$ for N times, and use these N rules to form the set $\mathbf{z}$.

Reasoning Predictor with Logic Rules. The reasoning predictor defines $p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \mathbf{z})$. Basically, given a query $\mathbf{q}$, existing graph $\mathcal{G}$, and rule set $\mathbf{z}$, the reasoning predictor tries to predict the answer $\mathbf{a}$.

Following the idea in stochastic logic programming (Cussens, 2000), we employ a log-linear model for reasoning. Basically, for a query $\mathbf{q} = (h, \mathtt{r}, ?)$, each logic rule for relation $\mathtt{r}$ can find different grounding paths in the knowledge graph, leading to different candidate answers.

Let $\mathcal{A}$ denote the set of all the candidate answers which are discovered by logic rules in set $\mathbf{z}$. For each candidate answer $e \in \mathcal{A}$, we define the following function $\mathtt{score}_w$ to compute a score:

$$\begin{aligned}\mathtt{score}_w(e) &= \sum_{\mathit{rule} \in \mathbf{z}} \mathtt{score}_w(e|\mathit{rule}) \\ &= \sum_{\mathit{rule} \in \mathbf{z}} \sum_{\mathit{path} \in \mathcal{P}(h, \mathit{rule}, e)} u_w(\mathit{rule}) \cdot v_w(\mathit{path}),\end{aligned} \tag{4}$$

where $\mathcal{P}(h, \mathit{rule}, e)$ is the set of grounding paths which start at h and end at e following a *rule* (e.g., $\mathit{Alice} \xrightarrow{\mathtt{friend}} \mathit{Bob} \xrightarrow{\mathtt{hobby}} \mathit{Sing}$). $u_w(\mathit{rule})$ and $v_w(\mathit{path})$ are scalar weights of each *rule* and *path*.

For the scalar weight $u_w(\mathit{rule})$ of *rule*, we parameterize it with a lookup table over the logic rules generated by the rule generator. For the score $v_w(\mathit{path})$ of each specific *path*, we explore two methods for parameterization. One method always sets $v_w(\mathit{path}) = 1$, and the other one follows the idea in an embedding-based approach RotatE (Sun et al., 2019). Specifically, we introduce an embedding for each entity, and each relation is modeled as a rotation operator on entity embeddings. For each grounding *path* of *rule* starting from h to e, we apply the rotation operator defined by each body relation of *rule* to the embedding of h, and compute the similarity between the derived embedding and the embedding of e as $v_w(\mathit{path})$. For example, given a path $\mathit{Alice} \xrightarrow{\mathtt{friend}} \mathit{Bob} \xrightarrow{\mathtt{hobby}} \mathit{Sing}$, we rotate *Alice*'s embedding with the operators defined by $\mathtt{friend}$ and $\mathtt{hobby}$. Then we compute the similarity between the new embedding and embedding of *Sing* as $v_w(\mathit{path})$. See App. B for the details of the parameterization.

Once we have the score for each candidate answer, we can further define the distribution over candidate answers by using a softmax function as follows:

$$p_w(\mathbf{a} = e|\mathcal{G}, \mathbf{q}, \mathbf{z}) = \frac{\exp(\mathtt{score}_w(e))}{\sum_{e' \in \mathcal{A}} \exp(\mathtt{score}_w(e'))}. \tag{5}$$

2.3. Optimization

Next, we explain how we optimize the reasoning predictor and rule generator with logic rules to maximize the objective in Eq. (2). Specifically, an EM framework is developed for optimizing the parameters, alternating between an E-step and an M-step. In the E-step, the rule generator generates multiple logic rules, from which a few most important ones are identified via posterior inference, with prior from the rule generator and likelihood from the reasoning predictor. In the M-step, the rule generator and reasoning predictor are updated to be consistent with the important rules selected in the E-step.

E-step. The general goal of the E-step is to identify the most important rules from the generator.

Formally, for each data instance $(\mathcal{G}, \mathbf{q}, \mathbf{a})$, we start with sampling a set of logic rules $\hat{\mathbf{z}}$ from the rule generator, i.e., $\hat{\mathbf{z}} \sim p_\theta(\mathbf{z}|\mathbf{q})$. Then we try to infer a set of most important rules $\mathbf{z}_*$ from all the logic rules $\hat{\mathbf{z}}$ generated by the rule generator, i.e., $\mathbf{z}_* \subset \hat{\mathbf{z}}$. This can be achieved by posterior inference, where we look into the posterior distribution of $\mathbf{z}_*$, i.e., $p_{\theta,w}(\mathbf{z}_*|\mathcal{G}, \mathbf{q}, \mathbf{a}) \propto p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \mathbf{z}_*) p_\theta(\mathbf{z}_*|\mathbf{q})$, with the prior of $\mathbf{z}_*$ defined by the rule generator p_θ and likelihood for $\mathbf{z}_*$ coming from the reasoning predictor p_w. The posterior distribution combines knowledge from both the rule generator and the reasoning predictor, and hence the likely subset of important rules $\mathbf{z}_*$ can be naturally obtained by sampling from the posterior. However, exact inference is intractable, so we use an approximation algorithm, and the workflow is summarized as follows:

1. For each $\mathit{rule} \in \hat{\mathbf{z}}$, compute the following H score:

$$\begin{aligned}H(\mathit{rule}) = \mathtt{score}_w(t|\mathit{rule}) - \frac{1}{|\mathcal{A}|} \sum_{e \in \mathcal{A}} \mathtt{score}_w(e|\mathit{rule}) \\ + \log \mathrm{RNN}_\theta(\mathit{rule}|r),\end{aligned} \tag{6}$$

2. Form a sample $\hat{\mathbf{z}}_*$ with K logic rules independently sampled from $\hat{\mathbf{z}}$, where the probability of sampling each *rule* is computed as $\exp(H(\mathit{rule}))/\sum_{\mathit{rule}' \in \hat{\mathbf{z}}} \exp(H(\mathit{rule}'))$.

We prove it in the appendix.

M-step. Once we obtain a set of important rules $\hat{\mathbf{z}}_*$ for each data instance $(\mathcal{G}, \mathbf{q}, \mathbf{a})$ in the E-step, we further leverage those rules to update the rule generator and the reasoning predictor in the M-step.

Specifically, for each data instance $(\mathcal{G}, \mathbf{q}, \mathbf{a})$, we treat the corresponding rule set $\hat{\mathbf{z}}_*$ as part of the (now complete) training data, and update the rule generator by maximizing the log-likelihood of $\hat{\mathbf{z}}_*$:

$$\begin{aligned}\max_\theta \log p_\theta(\hat{\mathbf{z}}_*|\mathbf{q}) &= \sum_{\mathit{rule} \in \hat{\mathbf{z}}_*} \log p_\theta(\mathit{rule}|\mathbf{q}) + \text{const} \\ &= \sum_{\mathit{rule} \in \hat{\mathbf{z}}_*} \log \mathrm{RNN}_\theta(\mathit{rule}|r) + \text{const}.\end{aligned} \tag{7}$$

With the above objective, the knowledge from the reasoning

Table 1. Results of reasoning on FB15k-237 and WN18RR. H@K is in %. [*] means the numbers are taken from original papers. [†] means we rerun the methods with the same evaluation process.

Category	Algorithm	FB15k-237					WN18RR				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
No Rule Learning	TransE*	357	0.294	-	-	46.5	3384	0.226	-	-	50.1
	DistMult*	254	0.241	15.5	26.3	41.9	5110	0.43	39	44	49
	ComplEx*	339	0.247	15.8	27.5	42.8	5261	0.44	41	46	51
	ConvE*	244	0.325	23.7	35.6	50.1	4187	0.43	40	44	52
	RotatE*	**177**	0.338	24.1	37.5	**53.3**	**3340**	0.476	42.8	49.2	**57.1**
Rule Learning	PathRank	-	0.087	7.4	9.2	11.2	-	0.189	17.1	20.0	22.5
	NeuralLP†	-	0.237	17.3	25.9	36.1	-	0.381	36.8	38.6	40.8
	DRUM†	-	0.238	17.4	26.1	36.4	-	0.382	36.9	38.8	41.0
	NLIL*	-	0.25	-	-	32.4	-	-	-	-	-
	MINERVA*	-	0.293	21.7	32.9	45.6	-	0.415	38.2	43.3	48.0
	M-Walk*	-	0.232	16.5	24.3	-	-	0.437	41.4	44.5	-
RNNLogic	w/o emb.	538	0.288	20.8	31.5	44.5	7527	0.455	41.4	47.5	53.1
	with emb.	232	**0.344**	**25.2**	**38.0**	53.0	4615	**0.483**	**44.6**	**49.7**	55.8

predictor can be effectively distilled into the rule generator. In this way, the rule generator can be more effectively optimized, which allows the generator to focus only on important logic rules for exploitation, and thereby reduce the search space.

For the reasoning predictor p_w, ideally we should feed $\hat{\mathbf{z}}_*$ into p_w and update p_w to maximize the log-likelihood of the correct answer, i.e., $p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \hat{\mathbf{z}}_*)$. However, this can lead to a mismatch between distributions of logic rules uesd in training and testing. Specifically, during training the logic rules are from the posterior inference in E-step, whereas during testing logic rules come from the rule generator p_θ. This distribution mismatch can downgrade the reasoning performance after training. To solve the problem, we notice that after updating the rule generator p_θ with the selected rules $\hat{\mathbf{z}}_*$, the rule generator will assign most of the probability mass to rules in $\hat{\mathbf{z}}_*$. Therefore, we could naturally use rules sampled from the generator p_θ to update the reasoning predictor, allowing us to generate consistent logic rules during training and testing. Formally, the objective for p_w is given as follows:

$$\max_w \mathbb{E}_{p_\theta(\mathbf{z}|\mathbf{q})}[\log p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \mathbf{z})] \simeq \log p_w(\mathbf{a}|\mathcal{G}, \mathbf{q}, \hat{\mathbf{z}}) \quad (8)$$

where $\hat{\mathbf{z}}$ is a set of logic rules drawn from the rule generator p_θ, i.e., $\hat{\mathbf{z}} \sim p_\theta(\mathbf{z}|\mathbf{q})$.

3. Experiment

3.1. Compared Algorithms

For rule learning methods, we consider path ranking (Lao & Cohen, 2010), which linearly combines candidate rules for reasoning. We also consider NeuralLP (Yang et al., 2017), DRUM (Sadeghian et al., 2019) and NLIL (Yang & Song, 2020), which develop differentiable methods following a logic framework called TensorLog (Cohen et al., 2018). Besides, two methods based on reinforcement learning are compared, including MINERVA (Das et al., 2018) and M-Walk (Shen et al., 2018). We also compare against many state-of-the-art knowledge graph embedding methods, including TransE (Bordes et al., 2013), DistMult (Yang et al., 2015), ComplEx (Trouillon et al., 2016), ConvE (Dettmers et al., 2018) and RotatE (Sun et al., 2019). For our proposed RNNLogic, we consider two variants. One variant always assigns a constant score to different grounding paths in the reasoning predictor. In other words, $v_w(path)$ in Eq. (4) is fixed as 1, and we denote this variant as *w/o emb.*. The other variant introduces entity and relation embeddings to compute $v_w(path)$, and we denote the variant as *with emb.*.

3.2. Results

We present the results in Tab. 1.

We first compare RNNLogic with existing rule learning methods. For methods based on neural logic programming (NeuralLP, DRUM, NLIL), RNNLogic significantly outperforms them on all the datasets. The reason is that RNNLogic employs the rule generator to reduce search space, allowing the reasoning predictor to focus only on the most informative logic rules. RNNLogic also achieves better results than reinforcement learning methods (MINERVA, M-Walk) in most cases. The reason is that RNNLogic is optimized with the EM framework, in which the reasoning predictor is able to provide more useful feedback to the rule generator, and thus overcome the challenge of sparse reward.

We then compare RNNLogic against state-of-the-art embedding-based methods. For RNNLogic with embeddings in the reasoning predictor (*with emb.*), it outperforms most compared methods in most cases, and the reason is that RNNLogic is able to use logic rules to enhance reasoning performance. For RNNLogic without embedding (*w/o emb.*), it achieves comparable results to embedding-based methods, especially on WN18RR, Kinship and UMLS where the training triplets are sparse.

References

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2013.

Cohen, W. W., Yang, F., and Rivard Mazaitis, K. Tensorlog: Deep learning meets probabilistic databases. *Journal of Artificial Intelligence Research*, 2018.

Cussens, J. Stochastic logic programs: sampling, inference and applications. In *UAI*, 2000.

Das, R., Dhuliawala, S., Zaheer, M., Vilnis, L., Durugkar, I., Krishnamurthy, A., Smola, A., and McCallum, A. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2018.

Dettmers, T., Minervini, P., Stenetorp, P., and Riedel, S. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 1997.

Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Koller, D. and Pfeffer, A. Probabilistic frame-based systems. In *AAAI/IAAI*, 1998.

Lao, N. and Cohen, W. W. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 2010.

Lao, N., Mitchell, T., and Cohen, W. W. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011.

Lin, X. V., Socher, R., and Xiong, C. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, 2018.

Neal, R. M. and Hinton, G. E. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*. Springer, 1998.

Qu, M. and Tang, J. Probabilistic logic neural networks for reasoning. In *NeurIPS*, 2019.

Sadeghian, A., Armandpour, M., Ding, P., and Wang, D. Z. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

Shen, Y., Chen, J., Huang, P.-S., Guo, Y., and Gao, J. M-walk: Learning to walk over graphs using monte carlo tree search. In *NeurIPS*, 2018.

Simic, S. On a global upper bound for jensen's inequality. *Journal of mathematical analysis and applications*, 2008.

Simić, S. On a new converse of jensen's inequality. *Publications de l'Institut Mathematique*, 85(99):107–110, 2009.

Sun, Z., Deng, Z.-H., Nie, J.-Y., and Tang, J. Rotate: Knowledge graph embedding by relational rotation in complex space. *ICLR*, 2019.

Teru, K. K. and Hamilton, W. L. Inductive relation prediction on knowledge graphs. In *ICML*, 2020.

Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., and Bouchard, G. Complex embeddings for simple link prediction. In *ICML*, 2016.

Xiong, W., Hoang, T., and Wang, W. Y. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

Yang, B., Yih, W.-t., He, X., Gao, J., and Deng, L. Embedding entities and relations for learning and inference in knowledge bases. *ICLR*, 2015.

Yang, F., Yang, Z., and Cohen, W. W. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, 2017.

Yang, Y. and Song, L. Learn to explain efficiently via neural logic inductive learning. In *ICLR*, 2020.

Zhang, Y., Chen, X., Yang, Y., Ramamurthy, A., Li, B., Qi, Y., and Song, L. Efficient probabilistic logic reasoning with graph neural networks. In *ICLR*, 2020.

A. Analysis of E-step

Recall that in the E-step of the optimization algorithm, we aim to sample from the posterior distribution over rule set. However, directly sampling from the posterior distribution is intractable due to the complicated form of the posterior distribution. Therefore, we introduce an approximation algorithm. In this section, we present some theoretical analysis of the approximation algorithm. More specifically, in Section A.1, we present and prove a proposition, which tries to approximate the true posterior distribution with a simpler form. In Section A.2, we show how to perform sampling based on the approximation of the posterior distribution.

A.1. Approximation of the Posterior Distribution

Proposition *Consider a data instance $(\mathcal{G}, \mathbf{q}, \mathbf{a})$ with $\mathbf{q} = (h, r, ?)$ and $\mathbf{a} = t$. For a set of rules $\hat{\mathbf{z}}$ generated by the rule generator p_θ, we can compute the following score H for each rule*

$$H(\textit{rule}) = \left\{ \texttt{score}_w(t|\textit{rule}) - \frac{1}{|\mathcal{A}|}\sum_{e\in\mathcal{A}} \texttt{score}_w(e|\textit{rule}) \right\} + \log \mathrm{RNN}_\theta(\textit{rule}|r),$$

where $\mathcal{A}$ is the set of all candidate answers discovered by rules in $\hat{\mathbf{z}}$, $\texttt{score}_w(e|\textit{rule})$ is the score that each rule contributes to entity e as defined in Section 3, $\mathrm{RNN}_\theta(\textit{rule}|r)$ is the prior probability of rule computed by the rule generator. Suppose $s = \max_{e\in\mathcal{A}} |\texttt{score}_w(e)| < 1$. Then for a subset of rules $\mathbf{z}_* \subset \hat{\mathbf{z}}$ with $|\mathbf{z}_*| = K$, the log-probability $\log p_{\theta,w}(\mathbf{z}_*|\mathcal{G}, \mathbf{q}, \mathbf{a})$ could be approximated as follows:

$$\left| \log p_{\theta,w}(\mathbf{z}_*|\mathcal{G}, \mathbf{q}, \mathbf{a}) - \left(\sum_{\textit{rule}\in\mathbf{z}_*} H(\textit{rule}) + G(\mathbf{z}_*) + \text{const} \right) \right| \le s^2 + O(s^4)$$

where const *is a constant term that is independent from $\mathbf{z}_*$, $G(\mathbf{z}_*) = \log(K!/\prod_{\textit{rule}\in\hat{\mathbf{z}}} n_{\textit{rule}}!)$, with K being the given size of set $\mathbf{z}_*$ and $n_{\textit{rule}}$ being the number of times each rule appears in $\mathbf{z}_*$.*

Proof: We first rewrite the posterior probability as follows:

$$\begin{aligned}\log p_{\theta,w}(\mathbf{z}_*|\mathbf{g}, \mathbf{q}, \mathbf{a}) &= \log p_w(\mathbf{a}|\mathbf{g}, \mathbf{q}, \mathbf{z}_*) + \log p_\theta(\mathbf{z}_*|\mathbf{q}) + \text{const} \\ &= \log \frac{\exp(\texttt{score}_w(t))}{\sum_{e\in\mathcal{A}} \exp(\texttt{score}_w(e))} + \log \mathrm{Mu}(\mathbf{z}_*|K, \mathrm{RNN}_\theta(\cdot|r)) + \text{const},\end{aligned}$$

where const is a constant term which does not depend on the choice of $\mathbf{z}_*$, and $\mathrm{RNN}_\theta(\cdot|r)$ defines a probability distribution over all the composition-based logic rules. The probability mass function of the multinomial distribution $\mathrm{Mu}(\mathbf{z}_*|K, \mathrm{RNN}_\theta(\cdot|r))$ can then be written as below:

$$\mathrm{Mu}(\mathbf{z}_*|K, \mathbf{q}) = \frac{K!}{\prod_{\textit{rule}\in\hat{\mathbf{z}}} n_{\textit{rule}}!} \prod_{\textit{rule}\in\hat{\mathbf{z}}} \mathrm{RNN}_\theta(\textit{rule}|r)^{n_{\textit{rule}}},$$

where $n_{\textit{rule}}$ is the number of times a *rule* appears in $\mathbf{z}_*$. Based on that, the posterior probability can then be rewritten as follows:

$$\begin{aligned}&\log p_{\theta,w}(\mathbf{z}_*|\mathbf{g}, \mathbf{q}, \mathbf{a}) \\ =&\log \frac{\exp(\texttt{score}_w(t))}{\sum_{e\in\mathcal{A}} \exp(\texttt{score}_w(e))} + \log \frac{K!}{\prod_{\textit{rule}\in\hat{\mathbf{z}}} n_{\textit{rule}}!} + \log \prod_{\textit{rule}\in\hat{\mathbf{z}}} \mathrm{RNN}_\theta(\textit{rule}|r)^{n_{\textit{rule}}} + \text{const} \\ =&\log \frac{\exp(\texttt{score}_w(t))}{\sum_{e\in\mathcal{A}} \exp(\texttt{score}_w(e))} + G(\mathbf{z}_*) + \sum_{\textit{rule}\in\mathbf{z}_*} \log \mathrm{RNN}_\theta(\textit{rule}|r) + \text{const} \\ =&\texttt{score}_w(t) - \log \sum_{e\in\mathcal{A}} \exp(\texttt{score}_w(e)) + G(\mathbf{z}_*) + \sum_{\textit{rule}\in\mathbf{z}_*} \log \mathrm{RNN}_\theta(\textit{rule}|r) + \text{const}\end{aligned}$$

where $G(\mathbf{z}_*) = \log(K!/\prod_{\textit{rule}\in\hat{\mathbf{z}}} n_{\textit{rule}}!)$, with K being the given size of set $\mathbf{z}_*$ and $n_{\textit{rule}}$ being the number of times each *rule* appears in $\mathbf{z}_*$.

The term $\log \sum_{e\in\mathcal{A}} \exp(\texttt{score}_w(e))$ in the posterior probability is computationally too expensive, thus we approximate it using Lemma 1, which we prove at the end of this section.

Lemma 1. *Let $e \in \mathcal{A}$ be a finite set of entities, let $|\texttt{score}_w(e)| \leq s < 1$, and let $\texttt{score}_w$ be a function from entities to real numbers. Then the following inequalities hold:*

$$0 \leq \log \left(\sum_{e \in \mathcal{A}} \exp(\texttt{score}_w(e)) \right) - \left(\sum_{e \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \texttt{score}_w(e) + \log(|\mathcal{A}|) \right) \leq s^2 + O(s^4).$$

Hence, using the lemma we can get the following upper bound of the posterior probability:

$$\begin{aligned}
&\log p_{\theta,w}(\mathbf{z}_*|\mathbf{g}, \mathbf{q}, \mathbf{a}) \\
=&\texttt{score}_w(t) - \log \sum_{e \in \mathcal{A}} \exp(\texttt{score}_w(e)) + G(\mathbf{z}_*) + \sum_{\textit{rule} \in \mathbf{z}_*} \log \text{RNN}_\theta(\textit{rule}|r) + \text{const} \\
\leq&\texttt{score}_w(t) - \sum_{e \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \texttt{score}_w(e) + G(\mathbf{z}_*) + \sum_{\textit{rule} \in \mathbf{z}_*} \log \text{RNN}_\theta(\textit{rule}|r) + \text{const} \\
=& \sum_{\textit{rule} \in \mathbf{z}_*} H(\textit{rule}) + G(\mathbf{z}_*) + \text{const},
\end{aligned}$$

and also the following lower bound of the posterior probability:

$$\begin{aligned}
&\log p_{\theta,w}(\mathbf{z}_*|\mathbf{g}, \mathbf{q}, \mathbf{a}) \\
=&\texttt{score}_w(t) - \log \sum_{e \in \mathcal{A}} \exp(\texttt{score}_w(e)) + G(\mathbf{z}_*) + \sum_{\textit{rule} \in \mathbf{z}_*} \log \text{RNN}_\theta(\textit{rule}|r) + \text{const} \\
\geq&\texttt{score}_w(t) - \sum_{e \in \mathcal{A}} \frac{1}{|\mathcal{A}|} \texttt{score}_w(e) + G(\mathbf{z}_*) + \sum_{\textit{rule} \in \mathbf{z}_*} \log \text{RNN}_\theta(\textit{rule}|r) + \text{const} - s^2 - O(s^4) \\
=& \sum_{\textit{rule} \in \mathbf{z}_*} H(\textit{rule}) + G(\mathbf{z}_*) + \text{const} - s^2 - O(s^4),
\end{aligned}$$

where const is a constant term which does not depend on $\mathbf{z}_*$.

By combining the lower and the upper bound, we get:

$$\left| \log p_{\theta,w}(\mathbf{z}_*|\mathcal{G}, \mathbf{q}, \mathbf{a}) - \left(\sum_{\textit{rule} \in \mathbf{z}_*} H(\textit{rule}) + G(\mathbf{z}_*) + \text{const} \right) \right| \leq s^2 + O(s^4)$$

Thus, it only remains to prove Lemma 1 to complete the proof. We use Theorem A.1 from (Simic, 2008) as a starting point:

Theorem A.1. *Suppose that $\tilde{x} = \{x_i\}_{i=1}^n$ represents a finite sequence of real numbers belonging to a fixed closed interval $I = [a, b]$, $a < b$. If f is a convex function on I, then we have that:*

$$\frac{1}{n} \sum_{i=1}^n f(x_i) - f\left(\frac{1}{n} \sum_{i=1}^n x_i \right) \leq f(a) + f(b) - 2f\left(\frac{a+b}{2} \right).$$

As $(-\log)$ is a convex function and $\exp(\texttt{score}_w(e)) \in [\exp(-s), \exp(s)]$, Theorem A.1 gives us that:

$$\begin{aligned}
&- \frac{1}{|\mathcal{A}|} \sum_{e \in \mathcal{A}} \log\left(\exp(\texttt{score}_w(e))\right) + \log\left(\frac{1}{|\mathcal{A}|} \sum_{e \in \mathcal{A}} \exp(\texttt{score}_w(e)) \right) \\
&\leq -\log(\exp(-s)) - \log(\exp(s)) + 2\log\left(\frac{\exp(-s) + \exp(s)}{2} \right).
\end{aligned}$$

After some simplification, we get:

$$\begin{aligned}
&\log\left(\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right)\\
&\le \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e)+\log(|\mathcal{A}|)+2\log\left(\frac{\exp(-s)+\exp(s)}{2}\right)\\
&= \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e)+\log(|\mathcal{A}|)+2s-2\log 2+2\log(1+\exp(-2s))\\
&\le \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e)+\log(|\mathcal{A}|)+s^2+O(s^4),
\end{aligned} \tag{9}$$

where the last inequality is based on Taylor's series $\log(1+e^x)=\log 2+\frac{1}{2}x+\frac{1}{8}x^2+O(x^4)$ with $|x|<1$. On the other hand, according to the well-known Jensen's inequality, we have:

$$\log\left(\frac{1}{|\mathcal{A}|}\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right)\ge\frac{1}{|\mathcal{A}|}\sum_{e\in\mathcal{A}}\log\left(\exp(\texttt{score}_w(e))\right),$$

which implies:

$$\log\left(\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right)\ge\sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e)+\log(|\mathcal{A}|). \tag{10}$$

By combining Equation (9) and Equation (10), we obtain:

$$0\le\log\left(\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right)-\left(\sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e)+\log(|\mathcal{A}|)\right)\le s^2+O(s^4).$$

This completes the proof.

□.

A.2. Sampling Based on the Approximation of the True Posterior

Based on the above proposition, the log-posterior probability $\log p_{\theta,w}(\mathbf{z}_*|\mathcal{G},\mathbf{q},\mathbf{a})$ could be approximated by $(\sum_{\textit{rule}\in\mathbf{z}_*}H(\textit{rule})+G(\mathbf{z}_*)+\text{const})$, with const being a term that does not depend on $\mathbf{z}_*$. This implies that we could construct a distribution $\hat{p}(\mathbf{z}_*)\propto\exp(\sum_{\textit{rule}\in\mathbf{z}_*}H(\textit{rule})+G(\mathbf{z}_*))$ to approximate the true posterior, and draw samples from $\hat{p}$ as approximation to the real samples from the posterior.

It turns out that the distribution $\hat{p}(\mathbf{z}_*)$ is a multinomial distribution. To see that, we rewrite $\hat{p}(\mathbf{z}_*)$ as:

$$\begin{aligned}
\hat{p}(\mathbf{z}_*)&=\frac{1}{Z}\exp\left(\sum_{\textit{rule}\in\mathbf{z}_*}H(\textit{rule})+G(\mathbf{z}_*)\right)\\
&=\frac{1}{Z}\exp\left(G(\mathbf{z}_*)\right)\prod_{\textit{rule}\in\mathbf{z}_*}\exp\left(H(\textit{rule})\right)\\
&=\frac{1}{Z}\frac{K!}{\prod_{\textit{rule}\in\hat{\mathbf{z}}}n_{\textit{rule}}!}\prod_{\textit{rule}\in\hat{\mathbf{z}}}\exp\left(H(\textit{rule})\right)^{n_{\textit{rule}}}\\
&=\frac{1}{Z'}\frac{K!}{\prod_{\textit{rule}\in\hat{\mathbf{z}}}n_{\textit{rule}}!}\prod_{\textit{rule}\in\hat{\mathbf{z}}}q(\textit{rule})^{n_{\textit{rule}}}\\
&=\frac{1}{Z'}\text{Mu}(\mathbf{z}_*|K,q),
\end{aligned}$$

where $n_{\textit{rule}}$ is the number of times a *rule* appears in the set $\mathbf{z}_*$, q is a distribution over all the generated logic rules $\hat{\mathbf{z}}$ with $q(\textit{rule})=\exp(H(\textit{rule}))/\sum_{\textit{rule}'\in\hat{\mathbf{z}}}\exp(H(\textit{rule}'))$, Z and Z' are normalization terms. By summing over $\mathbf{z}_*$ on both sides of the above equation, we obtain $Z'=1$, and thus we have:

$$\hat{p}(\mathbf{z}_*)=\text{Mu}(\mathbf{z}_*|K,q).$$

To sample from such a multinomial distribution, we could simply sample K rules independently from the distribution q, and form a sample $\hat{\mathbf{z}}_*$ with these K rules.

In practice, we observe that the hard-assignment EM algorithm (Koller & Pfeffer, 1998) works better than standard EM algorithm despite the reduced theoretical guarantees. In the hard-assignment EM algorithm, we need to draw a sample $\hat{\mathbf{z}}_*$ with the maximum posterior probability. Based on the above approximation $\hat{p}(\mathbf{z}_*)$ of the true posterior distribution $p_{\theta,w}(\mathbf{z}_*|\mathcal{G}, \mathbf{q}, \mathbf{a})$, we could simply construct such a sample $\hat{\mathbf{z}}_*$ with K rules which have the maximum probability under the distribution q. By definition, we have $q(\textit{rule}) \propto \exp(H(\textit{rule}))$, and hence drawing K rules with maximum probability under q is equivalent to choosing K rules with the maximum H values.