Université de Montréal

Towards Combining Deep Learning and Statistical Relational Learning for Reasoning on Graphs

par

Meng Qu

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée en vue de l'obtention du grade de
Philosophiæ Doctor (Ph.D.)
en Discipline

December 21, 2023

Université de Montréal

Faculté des arts et des sciences

Cette thèse intitulée

Towards Combining Deep Learning and Statistical Relational Learning for Reasoning on Graphs

présentée par

Meng Qu

a été évaluée par un jury composé des personnes suivantes :

Yoshua Bengio
(président-rapporteur)

Jian Tang
(directeur de recherche)

Aishwarya Agrawal
(membre du jury)

Hanghang Tong
(examinateur externe)

Florian Maire
(représentant du doyen de la FESP)

Résumé

Cette thèse se focalise sur l'analyse de données structurées en graphes, un format de données répandu dans le monde réel. Le raisonnement dans ces données est un enjeu clé en apprentissage automatique, avec des applications allant de la classification de nœuds à la prédiction de liens.

On distingue deux approches majeures pour le raisonnement dans les données en graphes : l'apprentissage relationnel statistique et l'apprentissage profond. L'apprentissage relationnel statistique construit des modèles graphiques probabilistes, efficaces pour capturer des dépendances complexes et intégrer des connaissances préexistantes, comme les règles logiques. Des méthodes notables incluent les réseaux logiques de Markov et les champs aléatoires conditionnels. L'apprentissage profond, quant à lui, se base sur l'apprentissage de représentations pertinentes des données observées pour une compréhension et un raisonnement rapides. Les réseaux neuronaux pour graphes (GNN) représentent un outil de pointe dans ce domaine.

La combinaison de l'apprentissage relationnel statistique et de l'apprentissage profond offre une perspective enrichie sur le raisonnement, promettant un cadre plus robuste et efficace. Cette thèse explore cette combinaison, en développant des méthodes qui intègrent les deux approches. L'apprentissage profond renforce l'efficacité de l'apprentissage et de l'inférence dans l'apprentissage relationnel statistique, tandis que ce dernier affine les prédictions de l'apprentissage profond.

Ce cadre intégré est appliqué à un éventail de tâches de raisonnement sur les graphes, démontrant son efficacité et ouvrant la voie à des recherches futures pour des cadres de raisonnement encore plus robustes.

Mots-clés : Raisonnement, Graphes, Apprentissage Profond, Apprentissage Relationnel Statistique.

Abstract

This thesis centers on the analysis of graph-structured data, a ubiquitous data format in the real world. Reasoning within graph-structured data has long been a fundamental problem in machine learning, with applications spanning from node classification to link prediction.

There are two principal approaches to tackle reasoning within graph-structured data: statistical relational learning and deep learning. Statistical relational learning techniques construct probabilistic graphical models based on observed data, excelling at capturing intricate dependencies of available evidence while accommodating prior knowledge, such as logic rules. Notable methods include Markov logic networks (MLNs) and conditional random fields (CRFs). In contrast, deep learning models harness the capability to learn meaningful representations from observed data, using these representations to rapidly comprehend and reason over the data. Graph neural networks (GNNs) have emerged as prominent tools in the realm of deep learning, achieving state-of-the-art results across a spectrum of tasks.

Statistical relational learning and deep learning offer distinct perspectives on reasoning. Intuitively, combining these paradigms promises to create a more robust framework that inherits expressive power, efficiency, and the ability to model joint dependencies while simultaneously acquiring representations for more effective reasoning. In pursuit of this vision, this thesis explores the concept, developing methods that seamlessly integrate deep learning and statistical relational learning. Specifically, deep learning enhances the efficiency of learning and inference within statistical relational learning, while statistical relational learning, in turn, refines the predictions generated by deep learning to improve the accuracy.

This integrated paradigm is applied across a diverse range of reasoning tasks on graphs. Empirical results demonstrate the effectiveness of this paradigm, encouraging further exploration to yield more robust reasoning frameworks.

Keywords: Reasoning, Graphs, Deep Learning, Statistical Relational Learning.

Contents

List of Tables

List of Figures

List of acronyms and abbreviations

GNN	Graph Neural Networks
GCN	Graph Convolutional Networks
GAT	Graph Attention Networks
KGE	Knowledge Graph Embedding
PLM	Pre-trained Language Models
SRL	Statistical Relational Learning
CRF	Conditional Random Fields
RMN	Relational Markov Networks
MLN	Markov Logic Networks
EM	Expectation Maximization

ELBO	Evidence Lower Bound
GMNN	Graph Markov Neural Networks
SPN	Structured Proxy Networks

Acknowledgements

I owe a debt of gratitude to numerous individuals who played pivotal roles in my research endeavors and provided unwavering support throughout the past five years.

My greatest thanks should go to my esteemed advisor, Dr. Jian Tang. My association with Jian spans an impressive decade, commencing during my undergraduate studies and culminating with the completion of my PhD. During this remarkable journey, I received invaluable insights from Jian on research. These lessons encompassed the art of comprehensive literature surveys, the cultivation of novel ideas, the design of experiments, and the skill of writing research papers. Crucially, Jian consistently upheld high standards for my research, targeting at publishing high-impact papers in esteemed venues. He always underscored the importance of setting ambitious goals. As he said, merely adhering to easily attainable goals might reduce the motivation to strive for further progress once they were reached. Conversely, by establishing and diligently pursuing rigorous, high-reaching goals, students could surmount obstacles and attain a distinguished level of research excellence. While I may not have fully grasped the wisdom behind this approach at the time, I gradually realize the profound impact of the high standards Jian instilled in me as the end of my PhD journey is approaching. Besides, Jian instilled in me the importance of thinking from a high-level perspective and cultivating the vision to identify the most important problems to solve. His skill in this regard has left a lasting impact on my own research interests. Under his guidance, I am constantly working to focus on important emerging issues, like deep learning, graph embedding, graph machine learning, and drug discovery. His visionary guidance still influences how I conduct my research, and I am still trying to improve this essential skill as he initially advised. Thank you, Jian, for your profound impact on my academic journey.

I would like to express my heartfelt gratitude to my parents, Xiaokun Qu and Xiaochun Ma. As your only child, I have been incredibly lucky to receive your unwavering love and support since the day I was born. Both of you embody diligence and resilience in the face of challenges, never complaining about circumstances but instead, dedicating yourselves to overcoming challenges through hard work. Your kindness and willingness to help others have endeared you to everyone around you. Thankfully, you imparted these admirable qualities to

me, shaping my character along the way. You are always told me that you are proud of me, but I want to emphasize that I am equally proud of you. Your love, values, and resilience have been a constant source of inspiration in my life.

I would also like to extend my appreciation to my wife, Xiaofan Xing. Our journey together began in primary school, and I have always considered myself incredibly fortunate to have met you, my soulmate, in this lifetime. Life is anything but predictable; it presents us with numerous challenges, and facing them alone can be a painful experience. However, you have been a constant source of support, standing by me through every decision I have made. Thanks to your presence, my life has been illuminated, breaking the monotony with your unwavering companionship. Despite your consistent efforts to present your best self to me, I am keenly aware of the hardships you faced, especially during the last five years. Your resilience in the face of difficult times, coupled with your optimism, positivity, and courage, has left an indelible mark on my heart. Your unwavering attitude has been a continual source of inspiration, motivating me in the face of desperation.

There are also a lot of people I want to thank, who helped me in various ways. These include Yoshua Bengio, Zhaocheng Zhu, Sophie Xhonneux, Shengchao Liu, Huiyu Cai, Minghao Xu, Zuobai Zhang, Chence Shi, Weiping Song, Jie Fu, Min Lin.

Chapter 1

Introduction

1.1. Reasoning on Graph-structured Data

In today's technologically driven era, we inhabit a complex and intricately interconnected world where diverse objects are interlinked through a range of relationships, thereby facilitating the creation of vast graph-structured data.

For example, we see this kind of data in social networks, where people are connected through friendships, forming a large network of friends. Another example is in knowledge graphs, where different entities or concepts are linked together in various ways, helping to show a big picture of how different information relates to each other. These scenarios represent just a fragment of environments where graph-structured data finds its application, illustrating the ubiquitous nature of these interconnected structures in our modern landscape.

Reasoning on such graph-structured data has anchored itself as a pivotal focal point in the field of machine learning for an extended period. One application is node classification, where we try to sort individual points in the network into groups based on their connections and features. Another is link prediction, where we try to guess the possible connections that might exist, helping to find new relationships and add to what we know already. These applications show the big role that understanding and working with graph-structured data has played in machine learning, helping to make sense of our connected world.

In this thesis, we focus on these reasoning tasks on such graph-structured data.

1.2. Statistical Relational Learning and Deep Learning

Reasoning on graph-structured data has been extensively studied in the literature of statistical relational learning (SRL) and deep learning (DL).

Statistical relational learning [37], grounded in statistical and logical foundations, provides an interpretable framework which enables a detailed understanding of the patterns

learned and facilitating the integration of prior expert knowledge. The majority of statistical relational learning techniques leverage probabilistic graphical models, such as Bayesian networks and Markov networks, to encapsulate the interdependent relationships among connected objects for reasoning purposes [31, 69]. Additionally, some approaches harness the power of inductive logic programming, incorporating first-order logic to enhance the reasoning process [106]. However, despite their capacity to capture joint dependencies and incorporate logical structures, learning and inference in these methods remain hard due to high complexity of graph structures among objects. This inherent complexity imposes severe constraints on expressive powers and efficiency of statistical relational learning methods.

Another line of research is based on the recent advancements in deep learning [73], notably in the domain of graph neural networks (GNNs) [42, 61, 137]. Deep learning techniques have demonstrated remarkable powers in discerning intricate patterns and capturing non-linear correlations by acquiring valuable representations of entities, such as nodes within graphs, through highly expressive neural architectures. Typically, these methods are amenable to end-to-end training, resulting in robust efficiency gains. Because of their substantial model capacity and efficiency, deep learning approaches often yield state-of-the-art outcomes in various applications, including node classification and link prediction. Nevertheless, these approaches struggle to adequately capture interdependencies among disparate pieces of evidence, hindering their effectiveness in facilitating reasoning. Furthermore, their limited interpretability presents significant practical challenges.

1.3. Motivation of Combining Both Worlds

Deep learning and statistical relational learning methods inherently complement each other. Intuitively, drawing upon the analogy presented by Bengio [5], we can liken deep learning to the System 1 thought process in the human brain, as defined by Kahneman [54]. Deep learning operates akin to perception, characterized by speed and quick recognition of patterns. In contrast, statistical relational learning bears resemblance to the System 2 thought process, manifesting as a slower, more logical approach. In human cognition, the synergy between System 1 and System 2 thought facilitates more effective decision-making. Similarly, we envision that the integration of deep learning and statistical relational learning can culminate in a more potent framework.

Technically, deep learning has remarkable expressive power and proficiency in representation learning, while statistical relational learning excels in accommodating joint dependencies and logic rules. The fusion of these two paradigms is poised to yield a powerful framework. The framework marries efficiency with profound understanding, potentially yielding superior performance in downstream tasks, while preserving interpretability and logical coherence.

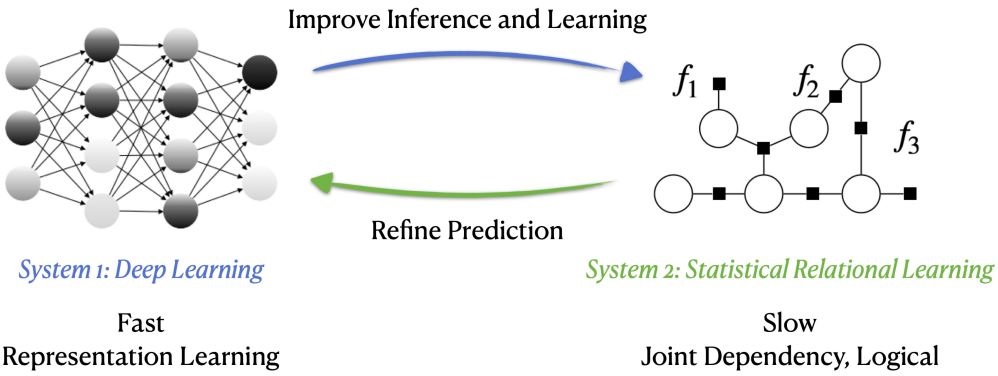


Figure 1. Framework overview. The framework presents a harmonious blend of two distinct paradigms: deep learning, which excels in fast reasoning through learned object representations, and statistical relational learning, which effectively captures the intricate joint dependencies of diverse evidence for more deliberate reasoning, in spite of a slower pace. This synergy seeks to harness the strengths of both worlds to enrich the reasoning process. Specifically, it involves leveraging the efficiency of deep learning to speed up the learning and inference phases of statistical relational learning, while simultaneously use statistical relational learning to finetune and enhance the predictions generated by deep learning methods.

Within the scope of this thesis, we endeavor to forge these synergies by developing hybrid approaches that seamlessly integrate statistical relational learning and deep learning for enhanced reasoning. The core concept is to encourage the their mutual enhancement during the reasoning process, resulting in performance improvements. Given the nontrivial nature of learning and inference in statistical relational learning, we leverage deep learning techniques to aid in both tasks, employing the idea of amortized inference. Simultaneously, we acknowledge that deep learning methods often struggle to effectively harness the joint dependencies within diverse evidence for reasoning. To address this limitation, we leverage statistical relational learning methods to refine and regularize the predictions generated by deep learning techniques, aligning them with the joint dependencies and logic specified by the statistical relational learning framework. This approach empowers deep learning methods to implicitly capture and model these dependencies. The overarching methodology is illustrated in Figure 1. Through this harmonious integration, we harness the strengths of both worlds, culminating in more potent reasoning approaches.

1.4. Outline

Next, we will delve into this paradigm in greater depth, elaborating on how it can be effectively employed across a variety of reasoning tasks centered on graph-structured data.

In Section 2, we explain how deep learning and statistical relational learning methods model graph-structured data in a probabilistic perspective. This probabilistic lens not only offers a deeper understanding of the rationale behind the fusion of deep learning and statistical relational learning but also serves to clarify the overall motivation.

Subsequently, we introduce a range of approaches tailored to address various reasoning tasks within the realm of graph-structured data. Our first focus lies on node classification, and we explore this challenge from both transductive and inductive perspectives. In Section 3, we unveil the Graph Markov Neural Network (GMNN), a novel approach that marries the relational Markov network from statistical relational learning with the graph convolutional network from deep learning. Moving forward to Section 4, we present another innovative method called Structured Proxy Networks (SPN). SPN integrates the principles of conditional random fields from statistical relational learning with the graph convolutional networks from deep learning. Both GMNN and SPN not only acquire valuable node representations but also model label dependencies between nodes, thereby elevating node classification performance.

The aforementioned approaches focus more on homogeneous graphs characterized by a single type of relation between nodes, yet real-world graphs often exhibit heterogeneity, resulting in multiple relations connecting nodes. One important example is the knowledge graph, where each edge is represented as a triplet (h, r, t), meaning that entity h relates to entity t via relation r. Within this context, a fundamental task is link prediction, aiming at predicting missing triplets. To tackle this challenge, we introduce the probabilistic Logic Neural Network (pLogicNet) in Section 5. pLogicNet represents a symbiotic union of Markov logic networks derived from statistical relational learning and knowledge graph embedding techniques inherent to deep learning. This integration empowers pLogicNet with the ability to adeptly deduce novel triplets. This ability is accomplished through the acquisition of entity embeddings, coupled with the utilization of predefined first-order logic rules (e.g., such as the rule that dictates "father's father is grandfather").

However, a potential limitation of pLogicNet arises from its reliance on a predefined set of high-quality logic rules for reasoning. To address this limitation, we delve into the realm of learning useful logic rules for knowledge graph reasoning. In Section 6, we propose a principled probabilistic approach called the RNNLogic. RNNLogic comprises a rule generator, parameterized by a recurrent neural network from deep learning, and a reasoning predictor, inspired by stochastic logic programming from statistical relational learning. These two modules undergo optimization via an EM-based algorithm, alternating between two major phases. During each iteration, the rule generator produces a set of logic rules, thereby enhancing the reasoning capabilities of the reasoning predictor. In the subsequent

phase, the reasoning predictor provides valuable feedback that refines the rule generator. Empirical experiments substantiate RNNLogic's capacity to autonomously acquire valuable rules conducive to effective reasoning on knowledge graphs.

Recognizing that the optimization of RNNLogic relies on an EM-based algorithm with associated approximations and a lack of end-to-end training, we further present DiffLogic in Section 7. DiffLogic extends the capabilities of RNNLogic and notably can be efficiently trained in an end-to-end manner.

Finally, in Section 9, we draw the thesis to a close, offering a summary of key findings, and identifying several promising avenues for future research.

1.5. Article Details

The thesis includes materials from a few papers where I am the first author or co-first author. The detailed list of these papers and my contribution are summarized as follows (* stands for equal contribution):

- **GMNN: Graph Markov Neural Networks.** Meng Qu, Yoshua Bengio, Jian Tang. *International Conference on Machine Learning, 2019.*

  Personal Contribution. I conceived the concept of GMNN, developed its mathematical formalization, and took charge of its implementation. I personally did all aspects of the experimental work and wrote the majority of the paper. Yoshua Bengio contributed by meticulously reviewing the formula, offering numerous valuable suggestions, and highlighting critical avenues for future research. Jian Tang played a pivotal role in shaping the overarching framework that integrates DL and SRL, which serves as the cornerstone of this thesis, and also played a role in refining the written content.
- **Neural structured prediction for inductive node classification.** Meng Qu*, Huiyu Cai*, Jian Tang. *International Conference on Learning Representations, 2022.*

  Personal Contribution. I conceptualized the idea and devised the mathematical formalization for SPN. I took the lead in developing the initial iteration of the model and was responsible for executing a portion of the experiments, as well as wrote the majority of the paper. Huiyu Cai actively contributed to the inception of the idea and played a vital role in enhancing the model precision and efficiency in its final iteration. Huiyu Cai also took charge of conducting the majority of the experiments. Jian Tang was instrumental in the initial idea generation, providing substantial feedback, and making significant improvements to the overall quality and clarity of the paper.
- **Probabilistic Logic Neural Networks for Reasoning.** Meng Qu, Jian Tang. *Advances in Neural Information Processing Systems, 2019.*

  Personal Contribution. I introduced the mathematical formalization for pLogicNet, led

the model implementation, managed all experimental procedures, and wrote the majority of the paper. Jian Tang played a pivotal role in problem formulation, participated in the initial idea generation, and contributed to refining the paper writing.

- **RNNLogic: Learning logic rules for reasoning on knowledge graphs.** Meng Qu*, Junkun Chen*, Louis-Pascal Xhonneux, Yoshua Bengio, Jian Tang. *International Conference on Learning Representations, 2021.*

  Personal Contribution. I initiated the concept and introduced the mathematical formalization of RNNLogic, let the implementation of the initial model version, conducted a portion of the experimental work, and composed the majority of the paper. Junkun Chen took charge of implementing the subsequent model version and conducted a portion of the experiments. Louis-Pascal Xhonneux actively participated in discussions related to the model and its mathematical formulation. Yoshua Bengio contributed by refining the mathematical formulation, offering crucial insights, and revising the paper writing. Jian Tang was responsible for posing the problem related to learning logic rules, played a role in refining the concepts, and made substantial improvements to the overall writing quality.
- **End-to-End Interpretable Logic Rule Learning for Knowledge Graph Reasoning.** Meng Qu, Jian Tang. *Ongoing.*

  Personal Contribution. I conceived the concept and formulated the mathematics behind DiffLogic, took the lead in implementing the model, managed all experimental aspects, and wrote the majority of the paper. Jian Tang played a collaborative role in brainstorming the conceptual ideas and contributed to enhancing the clarity and quality of the writing.

Furthermore, Section 8 provides a compelling glimpse into the diverse applications of our proposed paradigm. These examples are drawn from three articles in which I have contributed as the first author, co-first author, or second author. In that chapter, we will delve into the details of these articles, offering a comprehensive understanding of the content and author contributions.

Chapter 2

Preliminary

In this section, we rigorously define the challenge of reasoning on graph-structured data and present a formalization of this problem from a probabilistic standpoint. Within this probabilistic framework, we subsequently demonstrate how deep learning and statistical relational learning methods are employed for reasoning on graph-structured data respectively.

2.1. Problem Definition

This thesis is dedicated to the exploration of graph-structured data. Formally, we represent a graph as G, which inherently comprises a set of nodes denoted as V and a set of edges as E. Optionally, additional information such as node features and logic rules may be also available, collectively designated as I. Thus, we characterize a graph as $G = (V, E, I)$.

When dealing with such a graph, various reasoning tasks emerge, each aimed at inferring distinct types of knowledge embedded within the graph. For instance, node classification endeavors to predict labels for nodes, while link prediction seeks to anticipate missing edges within the graph. Concurrently, logic rule induction endeavors to uncover implicit logical rules from the graph. As these forms of knowledge remain unobserved, we can represent them collectively as a hidden variable, denoted by H.

Employing this notation, the problem of reasoning with graph-structured data can be reframed as the task of predicting H given G. Viewing it through a probabilistic lens, this problem revolves around modeling the distribution of H conditioned on G, i.e., $p(H|G)$.

2.2. Statistical Relational Learning

In the realm of literature, statistical relational learning has emerged as a powerful tool for modeling graph-structured data. Various models have been introduced, including relational Markov networks (RMN) [129], Markov logic networks (MLN) [106], and conditional random fields (CRF) [69]. In subsequent sections, we will delve into these approaches in details.

Fundamentally, these methods tackle the modeling of the joint distribution $p(H, G)$ using probabilistic graphical models, which encompass Bayesian networks and Markov networks. Through this framework, statistical relational learning adeptly captures the interdependency among the observed evidence G and the concealed knowledge H, enabling effective reasoning.

To acquire the joint distribution $p(H, G)$, we aim to maximize the log-likelihood of the observed graph, i.e., $\log p(G)$. However, optimizing the log-likelihood directly poses challenges due to the presence of hidden variables in H. Typically, the solution lies in the application of the EM algorithm, which leverages the following evidence lower bound (ELBO):

$$\log p(G) \geq \mathbb{E}_{q(H)}[\log p(H, G)] - \mathbb{E}_{q(H)}[\log q(H)]. \tag{2.2.1}$$

Here, $q(H)$ is a variable distribution, and the equation holds true only when the variational distribution equates to the ground truth posterior distribution $p(H|G)$, meaning $q(H) = p(H|G)$. The ELBO optimization unfolds through an iterative process involving an E-step and an M-step. In the E-step, we refine the variational distribution q to approximate the true posterior distribution $p(H|G)$. In the M-step, we fine-tune the joint distribution p to maximize the expected log-likelihood $\mathbb{E}_{q(H)}[\log p(H, G)]$.

Once we have learned the joint distribution $p(H, G)$, the reasoning tasks transition into an inference problem within graphical models, i.e., inferring the posterior distribution $p(H|G)$.

Despite the capability of statistical relational learning to effectively capture the intricate relationship among observed evidence and concealed knowledge, both the learning and inference processes in this field present significant challenges. The crux of the challenge lies in the need to work with the posterior distribution $p(H|G)$. During the learning phase, our objective is to align the variational distribution closely with the posterior distribution, while during inference, we aim to directly deduce the posterior itself. However, the inherent complexity of graph-structured data leads to intricate interdependencies among G and H, making the computation of $p(H|G)$ exceedingly challenging.

2.3. Deep Learning

Over the past decade, deep learning has showcased remarkable success across a multitude of domains. In the context of modeling graph-structured data, graph neural networks have emerged as formidable tools, consistently achieving state-of-the-art performance across a variety of applications. Notable approaches include graph convolutional networks (GCN) [61], graph attention networks (GAT) [137], and message passing neural networks (MPNN) [42].

In contrast to statistical relational learning, which centers on modeling the joint distribution $p(H, G)$, deep learning methods directly focus on characterizing the conditional

distribution $p(H|G)$. Similar to their statistical relational counterparts, deep learning encounters intricate interdependencies within the hidden knowledge H, thereby presenting unique challenges. To tackle this complexity, deep learning methods typically resort to the mean-field assumption, positing that different elements of H are conditionally independent given the graph G. Formally, this assumption translates to:

$$p(H|G) = \prod_{h \in H} p(h|G). \tag{2.3.1}$$

Consequently, the core task becomes modeling each term $p(h|G)$, representing the distribution of each individual hidden knowledge conditioned on the observed graph. For this purpose, deep learning methods frequently employ graph neural networks, which leverage a message passing mechanism to encode the graph and subsequently model the distribution $p(h|G)$ based on these encodings.

The mean-field assumption brings deep learning models a significant advantage in terms of training and inference efficiency. Moreover, the incorporation of message-passing mechanisms and nonlinear architectures in deep learning models imparts a high degree of expressive power, resulting in impressive performance across a wide array of applications. However, it is worth noting that the mean-field assumption comes at the cost of neglecting the interplay between hidden knowledge elements, which may yield suboptimal outcomes in complex scenarios and preclude the incorporation of prior knowledge (e.g., logic rules) for regularization of the hidden knowledge.

2.4. DL and SRL Integration Blueprint

Up to this point, we have observed that deep learning excels at rapid inference but tends to disregard joint dependencies, while statistical relational learning models these dependencies comprehensively but operates at a slower pace.

Our innovative approach involves integrating both a deep learning model and a statistical relational learning model, harnessing the strengths of one to compensate for the weaknesses of the other, thus achieving a harmonious synergy. Concretely, the deep learning model, denoted as q_{DL}, characterizes a conditional distribution $q_{\text{DL}}(H|G)$ in a mean-field form, expressed as $q_{\text{DL}}(H|G) = \prod_{h \in H} q_{\text{DL}}(h|G)$. On the other hand, the statistical relational learning model, labeled as p_{SRL}, models a joint distribution $p_{\text{SRL}}(H, G)$.

During the training process, both models are simultaneously optimized. When optimizing the statistical relational learning model p_{SRL}, we leverage the deep learning model to infer the hidden variables, for example, $\hat{H} \sim q_{\text{DL}}(H|G)$, and subsequently update p_{SRL} to maximize the complete log-likelihood $\log p_{\text{SRL}}(\hat{H}, G)$. By harnessing the high capacity and efficiency of deep learning models for inferring hidden variables, the training process of the

statistical relational learning model becomes efficient and swift. Simultaneously, for the deep learning model q_{DL}, we endeavor to minimize the Kullback-Leibler (KL) divergence between the posterior distribution defined by the deep learning model and that defined by the statistical relational learning model, expressed as $\text{KL}(q_{\text{DL}}(H|G)||p_{\text{SRL}}(H|G))$. In this manner, the statistical relational learning model essentially refines the predictions of the deep learning model regarding the hidden knowledge H. Furthermore, the patterns of joint dependencies and logic rules captured by the statistical relational learning model are implicitly distilled into the deep learning model, resulting in enhanced performance. Theoretically, this training framework elevates the Evidence Lower Bound (ELBO) of the log-likelihood $p_{\text{SRL}}(H, G))$, rendering it theoretically sound.

Upon completion of training, both the deep learning model q_{DL} and the statistical relational learning model p_{SRL} can be employed for inferring the hidden knowledge H.

In the upcoming sections, we will explore several examples that follow this blueprint, demonstrating the integration of deep learning and statistical relational learning for the purpose of reasoning with graph-structured data.

Chapter 3

GMNN: Graph Markov Neural Networks

This paper studies semi-supervised object classification in relational data, which is a fundamental problem in relational data modeling. The problem has been extensively studied in the literature of both statistical relational learning (e.g. relational Markov networks) and graph neural networks (e.g. graph convolutional networks). Statistical relational learning methods can effectively model the dependency of object labels through conditional random fields for collective classification, whereas graph neural networks learn effective object representations for classification through end-to-end training. In this paper, we propose the Graph Markov Neural Network (GMNN) that combines the advantages of both worlds. A GMNN models the joint distribution of object labels with a conditional random field, which can be effectively trained with the variational EM algorithm. In the E-step, one graph neural network learns effective object representations for approximating the posterior distributions of object labels. In the M-step, another graph neural network is used to model the local label dependency. Experiments on object classification, link classification, and unsupervised node representation learning show that GMNN achieves state-of-the-art results.

3.1. Introduction

We live in an interconnected world, where entities are connected through various relations. For example, web pages are linked by hyperlinks; social media users are connected via friendship relations. Modeling such relational data is important in machine learning with applications such as entity classification [97], link prediction [128] and link classification [26].

Many of these applications can be boiled down to the fundamental problem of semi-supervised object classification [129]. Specifically, objects [1] are interconnected and associated

[1] In this paper, we will use "object" and "node" interchangeably to refer to entities in the graph, because they are different terminologies used in the litteratures of statistical relational learning and graph neural networks.

with some attributes. Given the labels of a few objects, the goal is to infer the labels of other objects. This problem has been extensively studied in the literature of statistical relational learning (SRL), which develops statistical methods to model relational data. Some representative methods include relational Markov networks (RMN) [127] and Markov logic networks (MLN) [106]. Generally, these methods model the dependency of object labels using conditional random fields [69]. Because of their effectiveness for modeling label dependencies, these methods achieve compelling results on semi-supervised object classification. However, several limitations still remain. (1) These methods typically define potential functions in conditional random fields as linear combinations of some hand-crafted feature functions, which are quite heuristic. Moreover, the capacity of such models is usually insufficient. (2) Due to the complexity of relational structures between objects, inferring the posterior distributions of object labels for unlabeled objects remains a challenging problem.

Another line of research is based on the recent progress of graph neural networks [42, 46, 61, 137]. Graph neural networks approach object classification by learning effective object representations with non-linear neural architectures, and the whole framework can be trained in an end-to-end fashion. For example, the graph convolutional network (GCN) [61] iteratively updates the representation of each object by combining its own representation and the representations of the surrounding objects. These approaches have been shown to achieve state-of-the-art performance because of their effectiveness in learning object representations on relational data. However, one critical limitation is that the labels of objects are independently predicted based on their representations. In other words, the joint dependency of object labels is ignored.

In this paper, we propose a new approach called the Graph Markov Neural Network (GMNN), which combines the advantages of both statistical relational learning and graph neural networks. A GMNN is able to learn effective object representations as well as model label dependency between different objects. Similar to SRL methods, a GMNN includes a conditional random field [69] to model the joint distribution of object labels conditioned on object attributes. This framework can be effectively and efficiently optimized with the variational EM framework [89], alternating between an inference procedure (E-step) and a learning procedure (M-step). In the learning procedure, instead of maximizing the likelihood function, the training procedure for GMNNs optimizes the pseudolikelihood function [6] and parameterizes the local conditional distributions of object labels with a graph neural network. Such a graph neural network can well model the dependency of object labels, and no hand-crafted potential functions are required. For inference, since exact inference is intractable, we use a mean-field approximation [95]. Inspired by the idea of amortized inference [36, 60], we further parameterize the posterior distributions of object labels with another graph neural

network, which is able to learn useful object representations for predicting object labels. With a graph neural network for inference, the number of parameters can be significantly reduced, and the statistical evidence can be shared across different objects in inference [60].

Our GMNN approach is very general. Though it is designed for object classification, it can be naturally applied to many other applications, such as unsupervised node representation learning and link classification. Experiment results show that GMNNs achieve state-of-the-art results on object classification and unsupervised node representation learning, as well as very competitive results on link classification.

3.2. Related Work

In the literature of statistical relational learning (SRL), a variety of methods have been proposed for semi-supervised node classification. The basic idea is to model label dependency with probabilistic graphical models. Many early methods [31, 38, 39, 65, 152] are built on top of directed graphical models. However, these methods can only handle acyclic dependencies among nodes, and their performance for prediction is usually limited. Due to such weaknesses, many later SRL methods employ Markov networks (e.g., conditional random fields [69]), and representative methods include relational Markov networks (RMN) [127] and Markov logic networks (MLN) [106, 119]. Though these methods are quite effective, they still suffer from several challenges. (1) Some hand-crafted feature functions are required for specifying the potential function, and the whole framework is designed as a log-linear model by combining different feature functions, so the capacity of such frameworks is quite limited. (2) Inference remains challenging due to the complicated relational structures among nodes. Our proposed GMNN method overcomes the above challenges by using two different graph neural networks, one for modeling the label dependency and another for approximating the posterior label distributions, and the approach can be effectively trained with the variational EM algorithm.

Another category of related work is graph-based semi-supervised classification. For example, the label propagation methods [169, 170] iteratively propagate the label of each node to its neighbors. However, these methods can only model the linear dependency of node labels, while in our approach a non-linear graph neural network is used to model label dependency, which has greater expressive power. Moreover, GMNNs can also learn useful node representations for predicting node labels.

Another closely related research area is that of graph neural networks [22, 42, 46, 61, 137], which can learn useful node representations for predicting node labels. Essentially, the node representations are learned by encoding local graph structures and node attributes, and

the whole framework can be trained in an end-to-end fashion. Because of their effectiveness in learning node representations, they achieve state-of-the-art results in node classification. However, existing methods usually ignore the dependency between node labels. With GMNNs, besides learning node representations, we also model the joint dependency of node labels by introducing conditional random fields.

There are also some recent studies [163] using graph neural networks for inference in probabilistic graphical models. Compared with their work, our work focuses on statistical relational learning while their work puts more emphasis on standard graphical models. Moreover, our work utilizes two graph neural networks for both inference and learning, while their work only uses one graph neural network for inference.

3.3. Preliminary

3.3.1. Problem Definition

The problem of semi-supervised node classification considers a graph $G = (V,E,\mathbf{x}_V)$, in which V is a set of nodes, E is a set of edges between nodes, and $\mathbf{x}_V$ stands for the attributes of all the nodes. The edges in E may have multiple types, which represent different relations among nodes. For simplicity, here we assume all edges belong to the same type. Given the labels $\mathbf{y}_L$ of a few labeled nodes $L \subset V$, the goal is to predict the labels $\mathbf{y}_U$ for the remaining unlabeled nodes $U = V \setminus L$.

This problem has been extensively studied in the literature of both statistical relation learning (SRL) and graph neural networks (GNN). Essentially, both types of methods aim to model the distribution of node labels conditioned on the node attributes and the graph structure, i.e. $p(\mathbf{y}_V|\mathbf{x}_V,E)$. Next, we introduce the general idea of both methods. For notation simplicity, we omit E in the following formulas.

3.3.2. Statistical Relational Learning

Most SRL methods model $p(\mathbf{y}_V|\mathbf{x}_V)$ with conditional random fields, which employ the following formulation:

$$p(\mathbf{y}_V|\mathbf{x}_V) = \frac{1}{Z(\mathbf{x}_V)} \prod_{(n_i,n_j)\in E} \psi_{i,j}(\mathbf{y}_{n_i}, \mathbf{y}_{n_j}, \mathbf{x}_V). \tag{3.3.1}$$

Here, (n_i,n_j) is an edge in graph G, and $\psi_{i,j}(\mathbf{y}_{n_i}, \mathbf{y}_{n_j}, \mathbf{x}_V)$ is the potential score defined on the edge. Typically, the potential score is computed as a linear combination of some hand-crafted feature functions, such as logical formulae.

With this formulation, predicting the labels for unlabeled nodes becomes an inference problem, i.e., inferring the posterior label distribution of the unlabeled nodes $p(\mathbf{y}_U|\mathbf{y}_L,\mathbf{x}_V)$.

Exact inference is usually infeasible due to the complicated structures between node labels. Therefore, some approximation inference methods are often utilized, such as loopy belief propagation [85].

3.3.3. Graph Neural Network

Different from SRL methods, GNN methods simply ignore the dependency of node labels and they focus on learning effective node representations for label prediction. Specifically, the joint distribution of labels is fully factorized as:

$$p(\mathbf{y}_V|\mathbf{x}_V) = \prod_{n \in V} p(\mathbf{y}_n|\mathbf{x}_V). \tag{3.3.2}$$

Based on the formulation, GNNs will infer the label distribution $p(\mathbf{y}_n|\mathbf{x}_V)$ for each node n independently. For each node n, GNNs predict the label in the following way:

$$\mathbf{h} = g(\mathbf{x}_V, E) \quad p(\mathbf{y}_n|\mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W\mathbf{h}_n)),$$

where $\mathbf{h} \in \mathbb{R}^{|V| \times d}$ is the representations of all the nodes, and $\mathbf{h}_n \in \mathbb{R}^d$ is the representation of node n. $W \in \mathbb{R}^{K \times d}$ is a linear transformation matrix, with d as the representation dimension and K as the number of label classes. Cat stands for categorical distributions. Basically, GNNs focus on learning a useful representation $\mathbf{h}_n$ for each node n. Specifically, each $\mathbf{h}_n$ is initialized as the attribute representation of node n. Then each $\mathbf{h}_n$ is iteratively updated according to its current value and the representations of n's neighbors, i.e. $\mathbf{h}_{\mathrm{NB}(n)}$. For the updating function, the graph convolutional layer (GC) [61] and the graph attention layer (GAT) [137] can be used, or in general the neural message passing layer [42] can be utilized. After multiple layers of update, the final node representations are fed into a linear softmax classifier for label prediction. The whole framework can be trained in an end-to-end fashion with a few labeled nodes.

3.4. Model

In this section, we introduce our approach called the Graph Markov Neural Network (GMNN) for semi-supervised node classification. The goal of GMNN is to combine the advantages of both the statistical relational learning methods and graph neural networks, such that we can learn useful objective representations for predicting node labels, as well as model the dependency between node labels. Specifically, GMNN models the joint distribution of node labels conditioned on node attributes, i.e. $p(\mathbf{y}_V|\mathbf{x}_V)$, by using a conditional random field, which is optimized with a pseudolikelihood variational EM framework. In the E-step, a graph neural network is used to learn node representations for label prediction. In the

M-step, another graph neural network is employed to model the local dependency of node labels. Next, we introduce the details of the GMNN approach.

3.4.1. Pseudolikelihood Variational EM

Following existing SRL methods, we use a conditional random field as in Equation (3.3.1) to model the joint distribution of node labels conditioned on node attributes, i.e. $p_\phi(\mathbf{y}_V|\mathbf{x}_V)$, where the potential is defined over each edge, and ϕ is the model parameters. For now, we ignore the specific formulation of the potential function, and we will discuss it later.

We learn the model parameters ϕ by maximizing the log-likelihood function of the observed node labels, i.e. $\log p_\phi(\mathbf{y}_L|\mathbf{x}_V)$. However, directly maximizing the log-likelihood function is difficult, since many node labels are unobserved. Therefore, we instead optimize the evidence lower bound (ELBO) of the log-likelihood function:

$$\log p_\phi(\mathbf{y}_L|\mathbf{x}_V) \geq \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_L,\mathbf{y}_U|\mathbf{x}_V) - \log q_\theta(\mathbf{y}_U|\mathbf{x}_V)], \tag{3.4.1}$$

where $q_\theta(\mathbf{y}_U|\mathbf{x}_V)$ is a variational distribution, and the equation holds when $q_\theta(\mathbf{y}_U|\mathbf{x}_V) = p_\phi(\mathbf{y}_U|\mathbf{y}_L,\mathbf{x}_V)$. According to the variational EM algorithm [89], such a lower bound can be optimized by alternating between a variational E-step and an M-step. In the variational E-step (a.k.a., inference procedure), the goal is to fix p_ϕ and update the variational distribution $q_\theta(\mathbf{y}_U|\mathbf{x}_V)$ to approximate the true posterior distribution $p_\phi(\mathbf{y}_U|\mathbf{y}_L,\mathbf{x}_V)$.

In the M-step (a.k.a., learning procedure), we fix q_θ and update p_ϕ to maximize the likelihood function below:

$$\ell(\phi) = \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_L,\mathbf{y}_U|\mathbf{x}_V)]. \tag{3.4.2}$$

However, directly optimizing the likelihood function can be difficult, as we have to deal with the partition function in p_ϕ. To avoid computing the partition function, we instead optimize the pseudolikelihood function [6] below:

$$\ell_{PL}(\phi) \triangleq \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n\in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{V\setminus n},\mathbf{x}_V)] = \mathbb{E}_{q_\theta(\mathbf{y}_U|\mathbf{x}_V)}[\sum_{n\in V} \log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)], \tag{3.4.3}$$

where $\mathrm{NB}(n)$ is the neighbor set of n, and the equation is based on the independence properties of $p_\phi(\mathbf{y}_V|\mathbf{x}_V)$ derived from its formulation, i.e. Equation (3.3.1). The pseudolikelihood approach is widely used for learning Markov networks [62, 106]. Next, we introduce the details of the inference and learning steps.

3.4.2. E-step: Inference Procesure

The inference step aims to compute the posterior distribution $p_\phi(\mathbf{y}_U|\mathbf{y}_L,\mathbf{x}_V)$. Due to the complicated relational structures between node labels, exact inference is intractable.

Therefore, we approximate it with another variational distribution $q_\theta(\mathbf{y}_U|\mathbf{x}_V)$. Specifically, we use the mean-field method [95], in which q_θ is formulated as:

$$q_\theta(\mathbf{y}_U|\mathbf{x}_V) = \prod_{n\in U} q_\theta(\mathbf{y}_n|\mathbf{x}_V). \tag{3.4.4}$$

Here, n is the index of unlabeled nodes. In the variational distribution, all node labels are assumed to be independent.

To model the distribution of each node label in q_θ, we follow the idea of amortized inference [36, 60], and parameterize $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ with a graph neural network (GNN), which learns effective node representations for label prediction:

$$q_\theta(\mathbf{y}_n|\mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W_\theta \mathbf{h}_{\theta,n})). \tag{3.4.5}$$

Specifically, $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ is formulated as a categorical distribution, and the probability of each class is calculated by a softmax classifier based on the node representation $\mathbf{h}_{\theta,n}$. The representation $\mathbf{h}_{\theta,n}$ is learned by a GNN model with the node attributes $\mathbf{x}_V$ as features, and θ as parameters. We denote the GNN model as GNN_θ. With GNN_θ, we can improve inference by learning useful representations of nodes from their attributes and local connections. Besides, by sharing GNN_θ across different nodes, we can significantly reduce the number of parameters required for inference, which is more efficient [60].

With the above mean-field formulation, if we fix distribution $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$, then the optimum of $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$, denoted by $q^*(\mathbf{y}_n|\mathbf{x}_V)$, is specified by the following condition:

$$\log q^*(\mathbf{y}_n|\mathbf{x}_V) = \mathbb{E}_{q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)] + \mathrm{const.} \tag{3.4.6}$$

See Section A.1 for the proof. Bases on that, for each node n, we optimize $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ with a method similar to Salakhutdinov and Larochelle [109]. More specifically, we start by using $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$ to compute $q^*(\mathbf{y}_n|\mathbf{x}_V)$, which is further treated as target to update $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$. Computing $q^*(\mathbf{y}_n|\mathbf{x}_V)$ in Equation (3.4.6) relies on computing the expectation with respect to $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$. We estimate the expectation by drawing a sample from $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$, yielding:

$$\mathbb{E}_{q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)] \simeq \log p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V). \tag{3.4.7}$$

In the above formula, $\hat{\mathbf{y}}_{\mathrm{NB}(n)} = \{\hat{\mathbf{y}}_{n'}\}_{n'\in\mathrm{NB}(n)}$ is defined as below. For each unlabeled neighbor n' of node n, we sample $\hat{\mathbf{y}}_{n'} \sim q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$, and for each labeled neighbor n' of node n, $\hat{\mathbf{y}}_{n'}$ is set as the ground-truth label. In practice, we find that using one sample from $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$ yields comparable results with multiple samples. Therefore, in the experiments, only one sample is used for efficiency purpose.

According to the Equation (3.4.6) and Equation (3.4.7), we can obtain an approximation for $q^*(\mathbf{y}_n|\mathbf{x}_V)$ as $q^*(\mathbf{y}_n|\mathbf{x}_V) \approx p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V)$. Therefore, we could instead treat

$p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V)$ as target, and minimize $\mathrm{KL}(p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V)||q_\theta(\mathbf{y}_n|\mathbf{x}_V))$. We further use a parallel update strategy [64] to speed up training, where we jointly optimize $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ for every unlabeled node n, yielding the objective as follows:

$$O_{\theta,U} = \sum_{n\in U} \mathbb{E}_{p_\phi(\mathbf{y}_n|\hat{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V)}[\log q_\theta(\mathbf{y}_n|\mathbf{x}_V)]. \tag{3.4.8}$$

Besides, we notice that q_θ can be also trained by predicting the labels for the labeled nodes. Therefore, we also let q_θ maximize the following supervised objective function:

$$O_{\theta,L} = \sum_{n\in L} \log q_\theta(\mathbf{y}_n|\mathbf{x}_V). \tag{3.4.9}$$

Here, $\mathbf{y}_n$ is the ground-truth label of n. By adding Equation (3.4.8) and Equation (3.4.9), we obtain the overall objective for optimizing θ:

$$O_\theta = O_{\theta,U} + O_{\theta,L}. \tag{3.4.10}$$

3.4.3. M-step: Learning Procedure

In the M-step, we seek to learn the parameter ϕ. More specifically, we will fix q_θ and further update p_ϕ to maximize Equation (3.4.3). With the objective function, we notice that only the conditional distribution $p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)$ is required for p_ϕ in both the inference and learning steps (Equation (3.4.8) and Equation (3.4.3)). Therefore, instead of defining the joint distribution of node labels $p_\phi(\mathbf{y}_V|\mathbf{x}_V)$ by specifying the potential function, we can simply focus on modeling the conditional distribution. Here, we parameterize the conditional distribution $p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)$ with another non-linear graph neural network model (GNN) because of its effectiveness:

$$p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W_\phi \mathbf{h}_{\phi,n})). \tag{3.4.11}$$

Here, the distribution of $\mathbf{y}_n$ is characterized by a softmax classifier, which takes the node representation $\mathbf{h}_{\phi,n}$ learned by a GNN model as features, and we denote the GNN as GNN_ϕ. When learning the node representation $\mathbf{h}_{\phi,n}$, GNN_ϕ treats all the labels $\mathbf{y}_{\mathrm{NB}(n)}$ surrounding the node n as features. Therefore, GNN_ϕ essentially models local dependencies of node labels. With the above formulation, we no longer require any hand-crafted feature functions.

The framework is related to the label propagation methods [169, 170], which also update each node label by combining the surrounding labels. However, these methods propagate labels in a fixed and linear way, whereas GNN_ϕ is in a learnable and non-linear way.

One notable thing is that when defining $p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)$, GNN_ϕ only uses the node labels $\mathbf{y}_{\mathrm{NB}(n)}$ surrounding the node n as features, but GNN_ϕ is flexible to incorporate other features. For example, we can follow existing SRL methods, and take both the surrounding node labels $\mathbf{y}_{\mathrm{NB}(n)}$ and surrounding attributes $\mathbf{x}_{\mathrm{NB}(n)}$ as features in GNN_ϕ. We will discuss this variant in our experiment (see Section 3.5.2).

Algorithm 1 Optimization Algorithm

Input: A graph G, some labeled nodes (L,$\mathbf{y}_L$).

Output: Node labels $\mathbf{y}_U$ for unlabeled nodes U.

Pre-train q_θ with $\mathbf{y}_L$ according to Equation (3.4.9).

while not converge **do**

⊡ *M-Step: Learning Procedure*

Annotate unlabeled nodes with q_θ. Denote the sampled labels as $\hat{\mathbf{y}}_U$.

Set $\hat{\mathbf{y}}_V = (\mathbf{y}_L,\hat{\mathbf{y}}_U)$ and update p_ϕ with Equation (3.4.12).

⊡ *E-Step: Inference Procedure*

Annotate unlabeled nodes with p_ϕ and $\hat{\mathbf{y}}_V$. Let the inferred distribution be $p_\phi(\mathbf{y}_U)$.

Update q_θ with Equation (3.4.8) and Equation (3.4.9) based on $p_\phi(\mathbf{y}_U)$ and $\mathbf{y}_L$.

end while

Classify each unlabeled node n based on $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$.

Another thing is that based on the overall formulation of p_ϕ, i.e. Equation (3.3.1), each node label $\mathbf{y}_n$ should only depend on its adjacent node labels $\mathbf{y}_{\mathrm{NB}(n)}$ and node attributes $\mathbf{x}_V$, which implies GNN_ϕ should not have more than one message passing layer. However, a common practice in the literature of graph neural networks is to use multiple message passing layers during training, which can well model the long-range dependency between different nodes. Therefore, we also explore using multiple message passing layers to capture such long-range dependency.

When optimizing p_ϕ to maximize Equation (3.4.3), we estimate the expectation in Equation (3.4.3) by drawing a sample from $q_\theta(\mathbf{y}_U|\mathbf{x}_V)$. More specifically, if n is an unlabeled node, then we sample $\hat{\mathbf{y}}_n \sim q_\theta(\mathbf{y}_n|\mathbf{x}_V)$, and otherwise we set $\hat{\mathbf{y}}_n$ as the ground-truth label. Afterwards, the parameter ϕ can be optimized by maximizing the following objective function:

$$O_\phi = \sum_{n\in V} \log p_\phi(\hat{\mathbf{y}}_n|\hat{\mathbf{y}}_{\mathrm{NB(n)}},\mathbf{x}_V). \tag{3.4.12}$$

3.4.4. Optimization

To optimize GMNN, we pre-train q_θ with the labeled nodes. Then we alternatively optimize p_ϕ and q_θ until convergence. Afterwards, both p_ϕ and q_θ can be used to classify unlabeled nodes. In practice, q_θ consistently outperforms p_ϕ, and thus we use q_θ to infer node labels by default. We summarize the detailed optimization algorithm in Algorithm 1.

Dataset	Task	# Nodes	# Edges	# Features	# Classes	# Training	# Validation	# Test
Cora	OC / NRL	2,708	5,429	1,433	7	140	500	1,000
Citeseer	OC / NRL	3,327	4,732	3,703	6	120	500	1,000
Pubmed	OC / NRL	19,717	44,338	500	3	60	500	1,000
Bitcoin Alpha	LC	3,783	24,186	3,783	2	100	500	3,221
Bitcoin OTC	LC	5,881	35,592	5,881	2	100	500	5,947

Table 1. Statistics of datasets used in GMNN.

Category	Algorithm	Cora	Citeseer	Pubmed
SSL	LP	74.2	56.3	71.6
SRL	PRM	77.0	63.4	68.3
	RMN	71.3	68.0	70.7
	MLN	74.6	68.0	75.3
GNN	Planetoid *	75.7	64.7	77.2
	GCN *	81.5	70.3	79.0
	GAT *	83.0	72.5	79.0
GMNN	W/o Attr. in p_ϕ	83.4	73.1	81.4
	With Attr. in p_ϕ	**83.7**	72.9	81.8
	Best results	**83.7**	**73.6**	**81.9**

Table 2. Results of GMNN for object classification.

Category	Algorithm	Cora	Citeseer	Pubmed
GNN	DeepWalk *	67.2	43.2	65.3
	DGI *	82.3	**71.8**	76.8
GMNN	With only q_θ	78.1	68.0	79.3
	With q_θ and p_ϕ	**82.8**	71.5	**81.6**

Table 3. Results of GMNN for unsupervised node representation learning.

3.5. Experiment

In this section, we evaluate the performance of GMNN on three tasks, including object classification, unsupervised node representation learning, and link classification.

3.5.1. Settings

Datasets.

- For object classification, we follow existing studies [61, 137, 159] and use three benchmark datasets from Sen et al. [115] for evaluation, including Cora, Citeseer, Pubmed. In each dataset, 20 objects from each class are treated as labeled objects, and we use the same data partition as in Yang et al. [159]. Accuracy is used as the evaluation metric.
- For unsupervised node representation learning, we also use the above three datasets, in which objects are treated as nodes. We learn node representations without using any

labeled nodes. To evaluate the learned representations, we follow Veličković et al. [138] and treat the representations as features to train a linear classifier on the labeled nodes. Then we classify the test nodes and report the accuracy. Note that we use the same data partition as in object classification.

- For link classification, we construct two datasets from the Bitcoin Alpha and the Bitcoin OTC datasets [66, 67] respectively. The datasets contain graphs between Bitcoin users, and the weight of a link represents the trust degree of connected users. We treat links with weights greater than 3 as positive instances, and links with weights less than -3 are treated as negative ones. Given a few labeled links, We try to classify the test links. As the positive and negative links are quite unbalanced, we report the F1 score.

The statistics of these datasets can be found in Table 1. Here, OC, NRL, LC represent object classification, node representation learning and link classification respectively.

Compared Algorithms.

- **GNN Methods**. For object classification and link classification, we mainly compare with the recently-proposed Graph Convolutional Network [61] and Graph Attention Network [137]. However, GAT cannot scale up to both datasets in the link classification task, so the performance is not reported. For unsupervised node representation learning, we compare with Deep Graph Infomax [138], which is the state-of-the-art method. Besides, we also compare with DeepWalk [97] and Planetoid [159].
- **SRL Methods**. For SRL methods, we compare with the Probabilistic Relational Model [65], the Relational Markov Network [127] and the Markov Logic Network [106]. In the PRM method, we assume that the label of an object depends on the attributes of itself, its adjacent objects, and the labels of its adjacent objects. For the RMN and MLN methods, we follow Taskar et al. [127] and use a logistic regression model locally for each object. This logistic regression model takes the attributes of each object and also those of its neighbors as features. Besides, we treat the labels of two linked objects as a clique template, which is the same as in Taskar et al. [127]. In RMN, a complete score table is employed for modeling label dependency, which maintains a potential score for every possible combination of object labels in a clique. In MLN, we simply use one indicator function in the potential function, and the indicator function judges whether the objects in a clique have the same label. Loop belief propagation [85] is used for approximation inference in RMN and MLN.
- **SSL Methods**. For methods under the category of graph-based semi-supervised classification, we choose the label propagation method [169] to compare with.

Parameter Settings.

- **Object Classification.** For GMNN, p_ϕ and q_θ are composed of two graph convolutional layers with 16 hidden units and ReLU activation [86], followed by the softmax function, as in Kipf and Welling [61]. We reweight each feature of a node to 1 if the original weight is greater than 0. Dropout [120] is applied to the network inputs with $p = 0.5$. We use the RMSProp optimizer [131] during training, with the initial learning rate as 0.05 and weight decay as 0.0005. In each iteration, both networks are trained for 100 epochs. The mean accuracy over 100 runs is reported in experiment.
- **Unsupervised Node Representation Learning.** For the GMNN approach, p_ϕ and q_θ are composed of two graph convolutional layers followed by a linear layer and the softmax function. The dimension of hidden layers is set as 512 for Cora and Citeseer, and 256 for Pubmed, which are the same as in Veličković et al. [138]. ReLU [86] is used as the activation function. Each node feature is reweighted to 1 if the original weight is larger than 0. We apply dropout [120] to the inputs of both networks with $p = 0.5$. The Adam SGD optimizer [58] is used for training, with initial learning rate as 0.1 and weight decay as 0.0005. We empirically train q_θ for 200 epoches during pre-training. Afterwards, we train both p_ϕ and q_θ for 2 iterations, with 100 epochs for each network per iteration. For the Pubmed dataset, we transform the raw node features into binary value since it can result in better performance. The mean accuracy over 50 runs is reported.
- **Link Classification.** The setting of GMNN in this task is similar as in object classification, with the following differences. The dimension of the hidden layers is set as 128. No weight decay and dropout are used. In each iteration, both networks are trained for 5 epochs with the Adam optimizer [58], and the learning rate is 0.01.

3.5.2. Results

1. Comparison with the Baseline Methods. The quantitative results on the three tasks are presented in Table 2, Table 3, and Table 4 respectively. Here, the results are in % and [*] means the results are taken from corresponding papers. For object classification, GMNN significantly outperforms all the SRL methods. The performance gain is from two folds. First, during inference, GMNN employs a GNN model, which can learn effective object representations to improve inference. Second, during learning, we model the local label dependency with another GNN, which is more effective compared with SRL methods. GMNN is also superior to the label propagation method, as GMNN is able to use object attributes and propagate labels in a non-linear way. Compared with GCN, which employs the same architecture as the inference network in GMNN, GMNN significantly outperforms GCN, and the performance gain mainly comes from the capability of modeling label dependencies. Besides, GMNN also outperforms GAT, but their performances are quite close. This is

because GAT utilizes a much more complicated architecture. Since GAT is less efficient, it is not used in GMNN, but we anticipate the results can be further improved by using GAT, and we leave it as future work. In addition, by incorporating the object attributes in the learning network p_ϕ, we further improve the performance, showing that GMNN is flexible and also effective to incorporate additional features in the learning network. For link classification, we obtain similar results.

For unsupervised node representation learning, GMNN achieves state-of-the-art results on the Cora and Pubmed datasets. The reason is that GMNN effectively models the smoothness of the neighbor distributions for different nodes with the p_ϕ network. Besides, the performance of GMNN is quite close to the performance in the semi-supervised setting (Table 2), showing that the learned representations are quite effective. We also compare with a variant without using the p_ϕ network (with only q_θ). In this case, we see that the performance drops significantly, showing the importance of using p_ϕ as a regularizer over the neighbor distributions.

Category	Algorithm	Bitcoin Alpha	Bitcoin OTC
SSL	LP	59.68	65.58
SRL	PRM	58.59	64.37
	RMN	59.56	65.59
	MLN	60.87	65.62
GNN	DeepWalk	62.71	63.20
	GCN	64.00	65.69
GMNN	W/o Attr. in p_ϕ	65.59	66.62
	With Attr. in p_ϕ	**65.86**	**66.83**

Table 4. Results of GMNN for link classification.

2. Analysis of the Amortized Inference. In GMNN, we employ amortized inference, and parameterize the posterior label distribution by using a GNN model. In this section, we thoroughly look into this strategy, and present some analysis in Table 5. Here, the variant "Non-amortized" simply models each $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ as a categorical distribution with independent parameters, and performs fix-point iteration (i.e. Equation (3.4.6)) to calculate the value. We see that the performance of this variant is very poor on all datasets. By parameterizing the posterior distribution as a neural network, which leverages the own attributes of each object for inference, the performance (see "1 Linear Layer") is significantly improved, but still not satisfactory. With several GC layers, we are able to incorporate the attributes from the surrounding neighbors for each object, yielding further significant improvement. The above observations prove the effectiveness of our strategy for inferring the posterior label distributions.

Architecture	Cora	Citeseer	Pubmed
Non-amortized	45.3	28.1	42.2
1 Linear Layer	55.8	57.5	69.8
1 GC Layer	72.9	67.6	71.8
2 GC Layers	83.4	73.1	81.4
3 GC Layers	82.0	70.6	80.7

Table 5. Analysis of amortized inference used in GMNN.

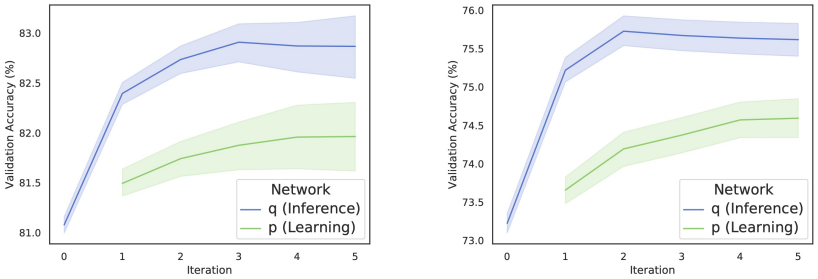
3. Ablation Study of the Learning Network. In GMNN, the conditional distribution $p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB(n)}},\mathbf{x}_V)$ is parameterized as another GNN, which essentially models the local label dependency. In this section, we compare different architectures of the GNN on the object classification task, and the results are presented in Table 6. Here, the variant "1 Mean Pooling Layer" computes the distribution of $\mathbf{y}_n$ as the linear combination of $\{\mathbf{y}_{n'}\}_{n'\in\mathrm{NB(n)}}$. This variant is similar to label propagation methods, and its performance is quite competitive. However, the weights of different neighbors during propagation are fixed. By parameterizing the conditional distribution with several GC layers, we are able to automatically learn the propagation weights, and thus obtain superior results on all datasets. This observation proves the effectiveness of employing GNNs in the learning procedure.

Architecture	Cora	Citeseer	Pubmed
1 Mean Pooling Layer	82.4	71.9	80.7
1 GC Layer	83.1	73.1	80.9
2 GC Layers	83.4	73.1	81.4
3 GC Layers	83.6	73.0	81.5

Table 6. Ablation study of the learning network in GMNN.

4. Convergence Analysis. In GMNN, we utilize the variational EM algorithm for optimization, which consists of an E-step and an M-step in each iteration. Next, we analyze the convergence of GMNN. We take the Cora and Citeseer datasets on object classification as examples, and report the validation accuracy of both the q_θ and p_ϕ networks at each iteration. Figure 2 presents the convergence curve, in which iteration 0 corresponds to the pre-training stage. GMNN takes only few iterations to convergence, which is very efficient.

5. Results on Random Data Splits. In the previous experiment, we have seen that GMNN significantly outperforms all the baseline methods for semi-supervised object classification under the data splits from Yang et al. [159]. To further validate the effectiveness of GMNN, we also evaluate GMNN on some random data splits. Specifically, we randomly create 10 data splits for each dataset. The size of the training, validation and test sets in each split is the same as the split in Yang et al. [159]. We compare GMNN with GCN [61]



(a) Cora (b) Citeseer

Figure 2. Convergence analysis of GMNN.

and GAT [137] on those random data splits, as they are the most competitive baseline methods. For each data split, we run each method with 10 different seeds, and report the overall mean accuracy in Table 7. We see GMNN consistently outperforms GCN and GAT on all datasets, proving the effectiveness of GMNN.

Algorithm	Cora	Citeseer	Pubmed
GCN	81.5	71.3	80.3
GAT	82.1	71.5	80.1
GMNN	**83.1**	**73.0**	**81.9**

Table 7. Object classification results of GMNN on random data splits.

6. Results in Few-shot Learning Settings. In the previous experiment, we have proved the effectiveness of GMNN for object classification in the semi-supervised setting. Next, we further conduct experiment in the few-short learning setting to evaluate the robustness of GMNN to data sparsity. We choose GCN and GAT for comparison. For each dataset, we randomly sample 5 labeled nodes under each class as training data, and run each method with 100 different seeds. The mean accuracy is shown in Table 8. We see GMNN significantly outperforms GCN and GAT. The improvement is even larger than the case of semi-supervised setting, where 20 labeled nodes under each class are used for training. This result proves that GMNN is robust to the sparsity of training data.

Algorithm	Cora	Citeseer	Pubmed
GCN	74.9	69.0	76.9
GAT	77.0	68.9	75.4
GMNN	**78.6**	**72.7**	**79.1**

Table 8. Object classification results of GMNN in few-shot learning settings.

7. Comparison with Self-training Methods. Our proposed GMNN approach is related to self-training frameworks. In GMNN, the p_ϕ network essentially tries to annotate unlabeled

objects, and the annotated objects are further treated as extra data to update q_θ through Equation (3.4.8). Similarly, in self-training, we typically use q_θ itself to annotate unlabeled objects, and collect extra training data for q_θ. Next, we compare GMNN with the self-training method for semi-supervised object classification, and the results are presented in Table 9.

We see GMNN consistently outperforms the self-training method. The reason is that the self-training method uses q_θ for both inference and annotation, while GMNN uses two different networks q_θ and p_ϕ to collaborate with each other. The information captured by q_θ and p_ϕ is complementary, and therefore GMNN achieves much better results.

Algorithm	**Cora**	**Citeseer**	**Pubmed**
Self-training	82.7	72.4	80.1
GMNN	**83.4**	**73.1**	**81.4**

Table 9. Comparison between GMNN and self-training methods.

8. Comparison of Different Approximation Methods. In GMNN, we use a mean-field variational distribution q_θ for inference, and the optimal q_θ is given by the fixed-point condition in Equation (3.4.6). Learning the optimal q_θ requires computing the right-hand side of Equation (3.4.6), which involves the expectation with respect to $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$ for each node n. To estimate the expectation, we notice that $q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)$ can be factorized as $\prod_{n'\in\mathrm{NB}(n)\cap U} q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$. Based on that, we can develop several empirical approximation methods.

Single Sample. The simplest way is to draw a single sample $\hat{\mathbf{y}}_{n'} \sim q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$ for each node $n' \in \mathrm{NB}(n) \cap U$, and then we could use the sample to estimate the expectation.

Multiple Samples. In practice, we can also draw multiple samples from $q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$ for each node $n' \in \mathrm{NB}(n) \cap U$ to estimate the expectation. Such a method has lower variance but entails higher cost.

Annealing. Another method is to introduce an annealing parameter τ in $q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$, so that we have:

$$q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V) = \mathrm{Cat}(\mathbf{y}_{n'}|\mathrm{softmax}(\frac{W_\theta \mathbf{h}_{\theta,n'}}{\tau})).$$

Then we can set τ to a small value (e.g. 0.1) and draw a sample $\hat{\mathbf{y}}_{n'} \sim q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$ for $n' \in \mathrm{NB}(n) \cap U$ to estimate the expectation, which typically has lower variance.

Max Pooling. Another method is max pooling, where we set $\hat{\mathbf{y}}_{n'} = \arg\max_{\mathbf{y}_{n'}} q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)$ for $n' \in \mathrm{NB}(n) \cap U$, and use $\{\hat{\mathbf{y}}_{n'}\}_{n'\in\mathrm{NB}(n)\cap U}$ as a sample to estimate the expectation.

Mean Pooling. Besides, we can also use the mean pooling method similar to the soft attention method in Deng et al. [25]. Specifically, suppose that we have C classes for classification, then the label of each node can be viewed as a one-hot C-dimensional vector. Based on that, we set $\bar{\mathbf{y}}_{n'} = \mathbb{E}_{q_\theta(\mathbf{y}_{n'}|\mathbf{x}_V)}[\mathbf{y}_{n'}]$ for each unlabeled neighbor n' of the object n, which can be understood as a soft label vector of that neighbor. For each labeled neighbor n' of the object n, we set $\bar{\mathbf{y}}_{n'}$ as the one-hot label vector. Then we can approximate the expectation as:

$$\mathbb{E}_{q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U}|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)},\mathbf{x}_V)] \approx p_\phi(\mathbf{y}_n|\bar{\mathbf{y}}_{\mathrm{NB}(n)},\mathbf{x}_V)] \stackrel{\mathrm{def}}{=} \mathrm{Cat}(\mathbf{y}_n|\mathrm{softmax}(W_\phi \mathbf{h}_{\phi,n})),$$

with $\mathbf{h}_{\phi,n} = g(\bar{\mathbf{y}}_{\mathrm{NB}(n)},E)$, where the object representation $\mathbf{h}_{\phi,n}$ is learned by feeding $\{\bar{\mathbf{y}}_{n'}\}_{n'\in\mathrm{NB}(n)}$ as features in a graph neural network g.

Comparison. We empirically compare different methods in the semi-supervised object classification task, where we use 10 samples for the multi-sample method and the parameter τ in the annealing method is set as 0.1. Table 10 presents the results. We see the annealing method consistently outperforms other methods on all datasets, and therefore we use the annealing method for all the experiments in the paper.

Method	Cora	Citeseer	Pubmed
Single Sample	82.1	71.5	80.4
Multiple Samples	83.2	72.5	81.1
Annealing	**83.4**	**73.1**	**81.4**
Max Pooling	83.2	72.8	81.2
Mean Pooling	**83.4**	72.6	80.5

Table 10. Comparison of different approximation methods for optimizing GMNN.

3.6. Conclusion

This paper studies semi-supervised object classification, which is a fundamental problem in relational data modeling, and a novel approach called the GMNN is proposed. GMNN employs a conditional random field to model the joint distribution of object labels, and two graph neural networks are utilized to improve both the inference and learning procedures. Experimental results on three tasks prove the effectiveness of GMNN. In the future, we plan to further improve GMNN to deal with graphs with multiple edge types, such as knowledge graphs [7].

Chapter 4

SPN: Structured Proxy Networks

This paper studies node classification in the inductive setting, i.e., aiming to learn a model on labeled training graphs and generalize it to infer node labels on unlabeled test graphs. This problem has been extensively studied with graph neural networks (GNNs) by learning effective node representations, as well as traditional structured prediction methods for modeling the structured output of node labels, e.g., conditional random fields (CRFs). In this paper, we present a new approach called the Structured Proxy Network (SPN), which combines the advantages of both worlds. SPN defines flexible potential functions of CRFs with GNNs. However, learning such a model is nontrivial as it involves optimizing a maximin game with high-cost inference. Inspired by the underlying connection between joint and marginal distributions defined by Markov networks, we propose to solve an approximate version of the optimization problem as a proxy, which yields a near-optimal solution, making learning more efficient. Extensive experiments on two settings show that our approach outperforms many competitive baselines [1].

4.1. Introduction

Graph-structured data are ubiquitous in the real world, covering a variety of applications. This paper studies node classification, a fundamental problem in the machine learning community. Most existing efforts focus on the transductive setting [61, 137], i.e., using a small set of labeled nodes in a graph to classify the rest of nodes. In this paper, we study node classification in the inductive setting [46], which is receiving growing interest. Given some training graphs with all nodes labeled, we aim to classify nodes in unlabeled test graphs.

This problem has been recently studied with graph neural networks (GNNs) [42, 46, 61, 137]. GNNs infer the marginal label distribution of each node by learning useful node representations based on node features and edges. Once a GNN is learned on training graphs,

[1] Codes are available at `https://github.com/DeepGraphLearning/SPN`.

it can be further applied to test graphs to infer node labels. Owing to the high capacity of nonlinear neural architectures, GNNs achieve impressive results on many datasets. However, one limitation of GNNs is that they ignore the joint dependency of node labels, and therefore node labels are predicted separately without modeling structured output.

Indeed, modeling structured output has been widely explored by the literature of structured prediction [3]. Structured prediction methods predict node labels collectively, so the label prediction of each node can be improved according to the predicted labels of neighboring nodes. One representative approach is the conditional random field (CRF) [69]. A CRF models the joint distribution of node labels with Markov networks, and thus training CRFs becomes a learning task in graphical models, while predicting node labels corresponds to an inference task. Typically, the potential functions in CRFs are parameterized as log-linear functions, which suffer from low model capacities. One remedy for this is to define potential functions with GNNs [80, 103]. However, most of the effective methods for learning CRFs involve a maximin game [125, 139], making learning often hard to converge, especially when GNNs are used to parameterize potential functions. Besides, as learning CRFs requires doing inference on the graphical models, the combined model requires a long run time.

In this paper, we address these challenges by proposing SPN (Structured Proxy Network), which is high in capacity, efficient in learning, and able to model the joint dependency of node labels. SPN is inspired by theoretical works in graphical models [139], which reveal close connections between the joint label distribution and the node/edge marginal label distribution in a Markov network. Based on that, we approximate the original optimization problem with a proxy problem, where the potential functions in CRFs are defined by combining a collection of node/edge pseudomarginal distributions, which are parameterized by GNNs that satisfy a few simple constraints. This proxy problem can be easily solved by maximizing the data likelihood on each node and edge, which yields a near-optimal joint label distribution on training graphs. Once the model is learned, we apply it to test graphs and run loopy belief propagation [85] to infer node labels. Experiments on two settings against both GNNs and CRFs prove the effectiveness of our approach.

Note that although SPN is tested on inductive node classification, this method is quite general and can be applied to many other structured prediction tasks as well, such as POS tagging [16] and named entity recognition [110]. Please refer to Section 4.4.3 for more details.

4.2. Related Work

Graph neural networks (GNNs) perform node classification by learning useful node representations [42, 61, 137]. Most earlier efforts focus on designing GNNs for transductive node classification [33, 151, 159], and many recent works move to the inductive setting [12, 15, 34, 46, 74, 164]. Because of high capacity and efficient training, GNNs achieve impressive results on inductive node classification. Despite the success, GNNs only try to model the marginal distribution of each node label and predict node labels separately without considering joint dependency. In contrast, SPN models joint distributions of node labels with CRFs, which predicts node labels collectively to improve results.

Another type of approach for inductive node classification is structured prediction, which focuses on modeling the dependency of node labels, so that the predicted node labels are more consistent. One representative approach is structured SVM [30, 111, 136], but it lacks a probabilistic interpretation to handle the uncertainty of the prediction. Another representative probabilistic approach is conditional random field [69, 123], which models the distribution of output spaces by using a Markov network. CRFs have been proven effective in many applications, such as POS tagging [69], shallow parsing [116], image labeling [48], and sequence labeling [70, 77, 81]. Nevertheless, the potential functions in CRFs are typically defined as log-linear functions, suffering from low model capacity.

There are also some recent works trying to combine GNNs and CRFs. Some works use GNNs to solve inference problems in graphical models [14, 22, 35, 112, 166]. In contrast, our approach uses GNNs to parameterize the potential functions in CRFs, which is in a similar vein to Ma et al. [78, 79, 80], Qu et al. [103], Wang et al. [141]. Among them, Ma et al. [80] and Qu et al. [103] optimize the pseudolikelihood [6] for model learning, and Wang et al. [141] optimizes a cross-entropy loss on each single node, which can yield poor approximation of the true joint likelihood [64, 125]. Our approach instead solves a proxy problem, which yields a near-optimal solution to the original problem of maximizing likelihood, and thus gets superior results. For Ma et al. [78] and Ma et al. [79], they focus on transductive node classification and continuous labels respectively, which are different from our work.

Lastly, learning CRFs has also been widely studied. Some works solve a maximin game as a surrogate for learning [125] and some others maximize a lower bound of the likelihood function [124]. However, these maximin games are often hard to optimize and the lower bounds are often loose. Different from them, we follow Wainwright et al. [140] and build an approximate optimization problem as a proxy, which is easier to solve and yields better results.

4.3. Preliminary

This paper focuses on inductive node classification [46], a fundamental problem in both graph machine learning and structured prediction. We employ a probabilistic formalization for the problem with some labeled training graphs and unlabeled test graphs. Each training graph is given as $(\mathbf{y}_V^*, \mathbf{x}_V, E)$, where $\mathbf{x}_V$ and $\mathbf{y}_V^*$ are features and labels of a set of nodes V, and E is a set of edges. For each test graph $(\mathbf{x}_{\tilde{V}}, \tilde{E})$, only features $\mathbf{x}_{\tilde{V}}$ and edges $\tilde{E}$ are given. Then we aim to solve:

- **Learning.** On training graphs, learn a probabilistic model to approximate $p(\mathbf{y}_V|\mathbf{x}_V, E)$.
- **Inference.** For each test graph, infer node labels $\mathbf{y}_{\tilde{V}}^*$ according to the distribution $p(\mathbf{y}_{\tilde{V}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$.

The problem has been extensively studied in both graph machine learning and structured prediction fields, and representative methods are GNNs and CRFs respectively. Next, we introduce the details.

4.3.1. Graph Neural Networks

For inductive node classification, graph neural networks (GNNs) learn node representations to predict marginal label distributions of nodes. GNNs assume all node labels are independent conditioned on node features and edges, so the joint label distribution is factorized into a set of marginals as below:

$$p_\theta(\mathbf{y}_V|\mathbf{x}_V, E) = \prod_{s \in V} p_\theta(y_s|\mathbf{x}_V, E). \tag{4.3.1}$$

Each marginal distribution $p_\theta(y_s|\mathbf{x}_V, E)$ is modeled as a categorical distribution over label candidates, and the label probabilities are computed by applying a linear softmax classifier to the representation of node s. In general, node representations are learned via the message passing mechanism [42], which brings high capacity to GNNs. Also, owing to the factorization in Equation (4.3.1), learning and inference can be easily solved in GNNs, where we simply need to compute loss and make prediction on each node separately. However, GNNs approximate only the marginal label distributions of nodes on training graphs, which may generalize badly and result in poor approximation of node marginal label distributions on test graphs. Also, the labels of different nodes are separately predicted according to their own marginal label distributions, yet the joint dependency of node labels is ignored.

4.3.2. Conditional Random Fields

For inductive node classification, conditional random fields (CRFs) build graphical models for node classification. A popular model is the pair-wise CRF, which formalizes the joint

label distribution as:

$$p_\theta(\mathbf{y}_V|\mathbf{x}_V, E) = \frac{1}{Z_\theta(\mathbf{x}_V, E)} \exp\{\sum_{s\in V} \theta_s(y_s,\mathbf{x}_V, E) + \sum_{(s,t)\in E} \theta_{st}(y_s, y_t,\mathbf{x}_V, E)\} \tag{4.3.2}$$

where $Z_\theta(\mathbf{x}_V, E)$ is the partition function. $\theta_s(y_s,\mathbf{x}_V, E)$ and $\theta_{st}(y_s, y_t,\mathbf{x}_V, E)$ are scalar scores contributed by each node s and each edge (s,t). In practice, these θ-functions can be either defined as simple linear functions or complicated GNNs. To make the notation concise, we will omit $\mathbf{x}_V$ and E in the θ-functions, e.g., simplifying $\theta_s(y_s,\mathbf{x}_V, E)$ as $\theta_s(y_s)$. With these θ-functions, CRFs are able to model the joint dependency of node labels and therefore achieve structured prediction.

However, learning CRFs to maximize likelihood $p_\theta(\mathbf{y}^*_V|\mathbf{x}_V, E)$ on training graphs is non-trivial in general, as the partition function $Z_\theta(\mathbf{x}_V, E)$ is typically intractable in graphs with loops. Thus, a major line of research instead optimizes a maximin game equivalent to likelihood maximization [139]. The maximin game for each training graph $(\mathbf{y}^*_V,\mathbf{x}_V, E)$ is formalized as follows:

$$\max_\theta \log p_\theta(\mathbf{y}^*_V|\mathbf{x}_V, E) = \max_\theta \min_q \mathcal{L}(\theta, q), \quad \text{with} \quad \mathcal{L}(\theta, q) = \\ \sum_{s\in V}\{\theta_s(y^*_s) - \mathbb{E}_{q_s(y_s)}[\theta_s(y_s)]\} + \sum_{(s,t)\in E}\{\theta_{st}(y^*_s, y^*_t) - \mathbb{E}_{q_{st}(y_s,y_t)}[\theta_{st}(y_s, y_t)]\} - H[q(\mathbf{y}_V)]. \tag{4.3.3}$$

Here, $q(\mathbf{y}_V)$ is a variational distribution on node labels, $q_s(y_s)$ and $q_{st}(y_s,y_t)$ are its marginal distributions on nodes and edges. $H[q(\mathbf{y}_V)] \coloneqq -\mathbb{E}_{q(\mathbf{y}_V)}[\log q(\mathbf{y}_V)]$ is the entropy of $q(\mathbf{y}_V)$. Given the maximin game, q and θ can be alternatively optimized via coordinate descent [125]. In each iteration, we first update the node and edge marginals $\{q_s(y_s)\}_{s\in V}$, $\{q_{st}(y_s, y_t)\}_{(s,t)\in E}$ towards those defined by p_θ. This can be done by MCMC, but the time cost is high, so approximate inference is often used, such as loopy belief propagation [85]. After q is optimized, we further update θ-functions with the node and edge marginals defined by q via gradient descent.

The optimal θ-functions are characterized by the following moment-matching conditions:

$$p_\theta(y_s|\mathbf{x}_V, E) = \mathbb{I}_{y^*_s}\{y_s\} \quad \forall s \in V, \qquad p_\theta(y_s,y_t|\mathbf{x}_V, E) = \mathbb{I}_{(y^*_s,y^*_t)}\{(y_s, y_t)\} \quad \forall (s,t) \in E, \tag{4.3.4}$$

where $\mathbb{I}_a\{b\}$ is an indicator function whose value is 1 if $a = b$ and 0 otherwise. See Section B.1 and Section B.2 in appendix for detailed derivation of the maximin game as well as the moment-matching conditions.

Once the θ-functions are learned, they can be further applied to each test graph $(\mathbf{x}_{\tilde{V}}, \tilde{E})$ to predict the joint label distribution as $p_\theta(\mathbf{y}_{\tilde{V}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$. Then the best label assignment $\mathbf{y}^*_{\tilde{V}}$ can be inferred by using approximate inference algorithms, such as loopy belief propagation [85].

The major challenge of CRFs lies in learning. On the one hand, learning relies on inference, meaning that we have to update $\{q_s(y_s)\}_{s\in V}$, $\{q_{st}(y_s, y_t)\}_{(s,t)\in E}$ to approximate the

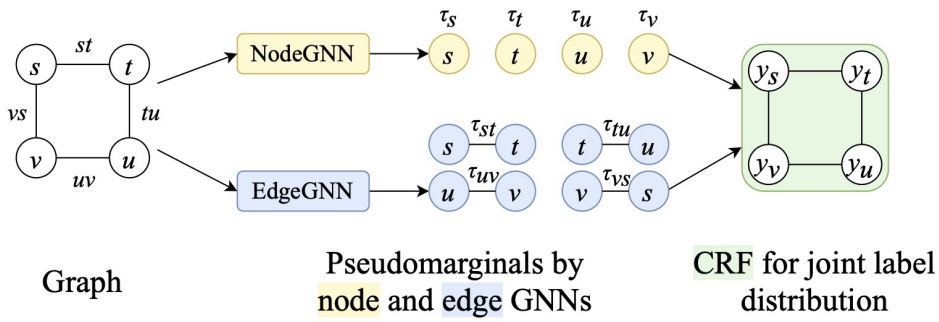


Figure 3. Overview of the SPN.

node and edge marginals of p_θ at each step, which can be expensive. On the other hand, as learning involves a maximin game and the optimal q of the inner minimization problem in Equation (4.3.3) is intractable, we can only maximize an upper bound of the likelihood function for θ, making learning unstable. The problem becomes even more severe when θ is parameterized by highly nonlinear neural models, e.g. GNNs.

4.4. Model

In this section, we introduce our proposed approach Structured Proxy Network (SPN). The general idea of SPN is to combine GNNs and CRFs by parameterizing potential functions in CRFs with GNNs, and therefore SPN enjoys high capacity and can model the joint dependency of node labels.

However, as elaborated in Section 4.3.2, learning such a model on training graphs is challenging due to the maximin game in optimization. Inspired by the connection between the joint and marginal distributions of CRFs, we instead construct a new optimization problem, which serves as a proxy for model learning. Compared with the original maximin game, the proxy problem is much easier to solve, where we can simply train two GNNs to approximate the marginal label distributions on nodes and edges, and further combine these pseudomarginals (defined in Proposition 1) into a near-optimal joint label distribution. This joint label distribution can be further refined by optimizing the maximin game, although it is optional and often unnecessary, as this distribution is often close enough to the optimal one. With this proxy problem for model learning, learning becomes more stable and efficient.

Afterwards, the learned model is used to predict the joint label distribution on test graphs. Then we run loopy belief propagation to infer node labels. Now, we introduce the details of our approach.

4.4.1. Learning

The learning task aims at training θ to maximize the log-likelihood function $\log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E)$ for each training graph $(\mathbf{y}_V^*,\mathbf{x}_V, E)$, which is highly challenging. Therefore, instead of directly optimizing this goal, we solve an approximate version of the problem as a proxy, which is training a node GNN and an edge GNN to maximize the log-likelihood of observed labels on nodes and edges.

The Proxy Problem. The proxy problem is inspired by Wainwright and Jordan [139], which points out that the marginal label distributions on nodes and edges defined by a Markov network have inherent connections with the joint distribution. This connection is stated in the proposition below.

Proposition 1. Consider a set of nonzero pseudomarginals $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(st)\in E}$ which satisfy $\sum_{y_s} \tau_{st}(y_s, y_t) = \tau_t(y_t)$ and $\sum_{y_t} \tau_{st}(y_s, y_t) = \tau_s(y_s)$ for all $(s,t) \in E$.

If we parameterize the θ-functions of p_θ in Equation (4.3.2) in the following way:

$$\theta_s(y_s) = \log \tau_s(y_s) \quad \forall s \in V, \qquad \theta_{st}(y_s, y_t) = \log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)\tau_t(y_t)} \quad \forall (s,t) \in E, \tag{4.4.1}$$

then $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$ are specified by a fixed point of the sum-product loopy belief propagation algorithm when applied to the joint distribution p_θ, which implies that:

$$\tau_s(y_s) \approx p_\theta(y_s) \quad \forall s \in V, \qquad \tau_{st}(y_s, y_t) \approx p_\theta(y_s, y_t) \quad \forall (s,t) \in E. \tag{4.4.2}$$

The proof is provided in Section B.3. With the proposition, we observe that if we parameterize the θ-functions by combining a set of pseudomarginals $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$ in the way defined by Equation (4.4.1), then those pseudomarginals can well approximate the true marginals of the joint distribution p_θ, i.e., $\tau_s(y_s) \approx p_\theta(y_s)$ and $\tau_{st}(y_s, y_t) \approx p_\theta(y_s,y_t)$ for all nodes s and edges (s, t). Given this precondition, if we further have $\tau_s(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ and $\tau_{st}(y_s, y_t) \approx \mathbb{I}_{(y_s^*,y_t^*)}\{(y_s,y_t)\}$, then the moment-matching conditions in Equation (4.3.4) for the optimal θ-functions are roughly satisfied. This implies the joint distribution $p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)$ derived in this way is a near-optimal one.

With the observation, rather than directly using GNNs to parameterize the θ-functions, we use a node GNN and an edge GNN to parameterize the pseudomarginals $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$. For the pseudomarginal $\tau_s(y_s)$ on node s, we apply the node GNN to node features $\mathbf{x}_V$ and edges E, yielding a representation $\mathbf{u}_s$ for node s. Then we apply a softmax classifier to $\mathbf{u}_s$ to compute $\tau_s(y_s)$:

$$\{\mathbf{u}_s\}_{u\in V} = \mathrm{GNN}_{\mathrm{node}}(\mathbf{x}_V, E), \qquad \tau_s(y_s) = \mathrm{softmax}(f(\mathbf{u}_s))[y_s], \tag{4.4.3}$$

where f maps a node representation to a $|\mathcal{Y}|$-dimensional logit and $\mathcal{Y}$ is the node label set. Similarly, we apply the edge GNN to compute a representation $\mathbf{v}_s$ for each node s, and model

$\tau_{st}(y_s, y_t)$ as:

$$\{\mathbf{v}_s\}_{s\in V} = \mathrm{GNN}_{\mathrm{edge}}(\mathbf{x}_V, E) \qquad \tau_{st}(y_s, y_t) = \mathrm{softmax}(g(\mathbf{v}_s, \mathbf{v}_t))[y_s, y_t], \tag{4.4.4}$$

where g is a function mapping a pair of representations to a $(|\mathcal{Y}| \times |\mathcal{Y}|)$-dimensional logit.

Given the parameterization, we construct the following problem as a proxy for learning θ-functions:

$$\begin{aligned} &\min_{\tau,\theta} \sum_{s\in V} d\left(\mathbb{I}_{y_s^*}\{y_s\}, \tau_s(y_s)\right) + \sum_{(s,t)\in E} d\left(\mathbb{I}_{(y_s^*, y_t^*)}\{(y_s,y_t)\}, \tau_{st}(y_s, y_t)\right), \\ &\text{subject to} \quad \theta_s = \log \tau_s(y_s), \quad \theta_{st}(y_s, y_t) = \log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)\tau_t(y_t)}, \\ &\text{and} \quad \sum_{y_s} \tau_{st}(y_s, y_t) = \tau_t(y_t), \quad \sum_{y_t} \tau_{st}(y_s, y_t) = \tau_s(y_s), \end{aligned} \tag{4.4.5}$$

for all nodes and edges, where d can be any divergence measure between two distributions. By solving the above problem, $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$ will be valid pseudo-marginals which can well approximate the true labels, i.e., $\tau_s(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ and $\tau_{st}(y_s, y_t) \approx \mathbb{I}_{(y_s^*, y_t^*)}\{(y_s,y_t)\}$. Then according to the constraint in the second line of Equation (4.4.5), θ-functions are formed in a way to enable $\tau_s(y_s) \approx p_\theta(y_s)$ and $\tau_{st}(y_s, y_t) \approx p_\theta(y_s,y_t)$ as stated in the Proposition 1. Combining these two sets of formula results in $p_\theta(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ and $p_\theta(y_s,y_t) \approx \mathbb{I}_{y_s^*}\{y_s\}$. We see that the moment-matching conditions in Equation (4.3.4) for the optimal joint label distribution are roughly achieved, implying that the derived joint distribution $p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)$ is a near-optimal solution to the original learning problem.

One good property of the proxy problem is that it can be solved easily. The last consistency constraint (i.e. $\sum_{y_s} \tau_{st}(y_s, y_t) = \tau_t(y_t)$ and $\sum_{y_t} \tau_{st}(y_s, y_t) = \tau_s(y_s)$) can be ignored during optimization, since by optimizing the objective function, the optimal pseudomarginals τ should well approximate the observed node and edge marginals, i.e., $\tau_s(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ and $\tau_{st}(y_s, y_t) \approx \mathbb{I}_{(y_s^*, y_t^*)}\{(y_s,y_t)\}$, and hence τ will almost naturally satisfy the consistency constraint. We also tried some constrained optimization methods to handle the consistency constraint, but they yield no improvement. See Section B.4 of appendix for more details. Thus, we can simply train the pseudomarginals parameterized by GNNs to approximate the true node and edge labels on training graphs, i.e., minimizing $d(\mathbb{I}_{y_s^*}\{y_s\}, \tau_s(y_s))$ and $d(\mathbb{I}_{(y_s^*, y_t^*)}\{(y_s,y_t)\}, \tau_{st}(y_s, y_t))$. Then we build θ-functions as in Equation (4.4.1) to obtain a near-optimal joint distribution. In practice, we choose d to be the KL divergence, yielding an objective for τ as:

$$\max_{\tau} \sum_{s\in V} \log \tau_s(y_s^*) + \sum_{(s,t)\in E} \log \tau_{st}(y_s^*, y_t^*). \tag{4.4.6}$$

This objective function is very intuitive, where we simply try to optimize the node GNN and edge GNN to maximize the log-likelihood function of the observed labels on nodes and edges.

Refinement. By solving the proxy problem, we can obtain a near-optimal joint distribution. In practice, we observe that when we have a large amount of training data, further refining this joint distribution by solving the maximin game in Equation (4.3.3) for a few iterations can lead to further improvement. Formally, each iteration of refinement has two steps. In the first step, we run sum-product loopy belief propagation [85], which yields a collection of node and edge marginals (i.e., $\{q_s(y_s)\}_{s\in V}$ and $\{q_{st}(y_s, y_t)\}_{(s,t)\in E}$) as approximation to the marginals defined by p_θ. In the second step, we update the θ-functions parameterized by the node and edge GNNs to maximize:

$$\sum_{s\in V}\Big\{\theta_s(y_s^*) - \mathbb{E}_{q_s(y_s)}[\theta_s(y_s)]\Big\} + \sum_{(s,t)\in E}\Big\{\theta_{st}(y_s^*, y_t^*) - \mathbb{E}_{q_{st}(y_s,y_t)}[\theta_{st}(y_s, y_t)]\Big\}. \tag{4.4.7}$$

Intuitively, we treat the true label y_s^* and (y_s^*, y_t^*) of each node and edge as positive examples, and encourage the θ-functions to raise up their scores. Meanwhile, those labels sampled from $q_s(y_s)$ and $q_{st}(y_s,y_t)$ act as negative examples, and the θ-functions are updated to decrease their scores.

4.4.2. Inference

After learning, we apply the node and edge GNNs to each test graph $(\mathbf{x}_{\tilde{V}}, \tilde{E})$ to compute the θ-functions, which are integrated into an approximate joint label distribution $p_\theta(\mathbf{y}_{\tilde{V}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$. Then we use this distribution to infer the best label $\mathbf{y}_{\tilde{s}}^*$ for each node $\tilde{s} \in \tilde{V}$, where two settings are considered.

Node-level Accuracy. Typically, we care about the node-level accuracy, i.e., how likely we can correctly classify a node in test graphs. Intuitively, the best label $y_{\tilde{s}}^*$ for each test node $\tilde{s} \in \tilde{V}$ should be predicted as $y_{\tilde{s}}^* = \arg\max_{y_{\tilde{s}}} p_\theta(y_{\tilde{s}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$, where $p_\theta(y_{\tilde{s}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$ is the marginal label distribution of node $\tilde{s}$ induced by the joint $p_\theta(\mathbf{y}_{\tilde{V}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$. In practice, the exact marginal is intractable, so we apply loopy belief propagation [85] for approximate inference. For each edge $(\tilde{s}, \tilde{t})$ in test graphs, we introduce a message function $m_{\tilde{t}\to\tilde{s}}(y_{\tilde{s}})$ and iteratively update all messages as:

$$m_{\tilde{t}\to\tilde{s}}(y_{\tilde{s}}) \propto \sum_{y_{\tilde{t}}}\{\exp(\theta_{\tilde{t}}(y_{\tilde{t}}) + \theta_{\tilde{s}\tilde{t}}(y_{\tilde{s}}, y_{\tilde{t}})) \prod_{\tilde{s}'\in N(\tilde{t})\setminus\tilde{s}} m_{\tilde{s}'\to\tilde{t}}(y_{\tilde{t}})\}, \tag{4.4.8}$$

where $N(\tilde{s})$ denotes the set of neighboring nodes for node $\tilde{s}$. Once the above process converges or after sufficient iterations, the label of each node $\tilde{s}$ can be inferred in the following way:

$$y_{\tilde{s}}^* = \arg\max_{y_{\tilde{s}}}[\exp(\theta_{\tilde{s}}(y_{\tilde{s}})) \prod_{\tilde{t}\in N(\tilde{s})} m_{\tilde{t}\to\tilde{s}}(y_{\tilde{s}})]. \tag{4.4.9}$$

Graph-level Accuracy. In some other cases, we might care about the graph-level accuracy, i.e., how likely we can correctly classify all nodes in a given test graph. In this case, the best prediction of node labels is given by $\mathbf{y}_{\tilde{V}}^* = \arg\max_{\mathbf{y}_{\tilde{V}}} p(\mathbf{y}_{\tilde{V}}|\mathbf{x}_{\tilde{V}}, \tilde{E})$. This problem can be

approximately solved by the max-product variant of loopy belief propagation, which simply replaces the sum over $y_{\bar{t}}$ in Equation (4.4.8) with max [148]. Afterwards, the best node label can be still decoded via Equation (4.4.9).

4.4.3. Discussion

In practice, many structured prediction problems can be viewed as special cases of inductive node classification, where the graphs between nodes have some special structures. For example in sequence labeling tasks (e.g., named entity recognition), the graphs between nodes have sequential structures. Thus, SPN can be applied to these tasks as well. In order for better results, one might replace GNNs with other neural models which are specifically designed for the studied task to better estimate the pseudomarginals. For example in sequence labeling tasks, recurrent neural networks can be used.

4.5. Experiment

4.5.1. Settings

Datasets. We consider datasets in two settings, which focus on node-level and graph-level accuracy respectively.

- **Node-level Accuracy.** The node-level accuracy measures *how likely a model can predict the correct label of a node in test graphs*. We use the **PPI** dataset [46, 172], which has 20 training graphs. To make the dataset more challenging, we also try using only the first 1/2/10 training graphs, yielding another three datasets **PPI-1**, **PPI-2**, and **PPI-10**. Besides, we also build a **DBLP** dataset from the citation network in Tang et al. [126]. Papers from eight conferences are treated as nodes, and we split them into three categories for classification according to conference domains [2]. For each paper, we compute the mean GloVe embedding [96] of words in the title and abstract as node features. The training/validation/test graph is formed as the citation graph of papers published before 1999, from 2000 to 2009, after 2010 respectively.
- **Graph-level Accuracy.** The graph-level accuracy measures *how likely a model can correctly classify all the nodes for a given test graph*. We construct three datasets from the Cora, Citeseer, and Pubmed datasets used for transductive node classification [159]. Each raw dataset has a single graph. For each training/validation/test node of the raw dataset, we treat its ego network [3] as a training/validation/test graph. We denote the datasets as **Cora***, **Citeseer***, **Pubmed***.

[2] ML: ICML/NeurIPS. CV: ICCV/CVPR/ECCV. NLP: ACL/EMNLP/NAACL.

[3] The local subgraph formed by a node and its direct neighbors.

Compared Algorithms. We choose the following methods for comparison, including a few well-known graph neural networks, conditional random fields, and our approach.

- **Graph Neural Networks.** For GNNs, we choose a few well-known model architectures for comparison, including GCN [61], GraphSage [46], GAT [137], Graph U-Net [33] and GCNII [12].
- **Conditional Random Fields.** For CRFs, we consider three variants. (1) *CRF-linear.* This variant uses linear θ-functions in Equation (4.3.2), which takes the features on nodes and edges for computation. (2) *CRF-GNN.* This variant parameterizes the θ-functions as $\theta_s(y_s) = f(\mathbf{u}_s)$ and $\theta_{st}(y_s,y_t) = g(\mathbf{v}_s, \mathbf{v}_t)$, with f and g defined in Equation (4.4.3) and Equation (4.4.4), where the node representations are generated by different GNN architectures (e.g., CRF-GAT). We train these models via the maximin game as in Equation (4.3.3) with sum-product loopy belief propagation. (3) *GMNN.* We also consider GMNN [103], an approach combining GNNs and CRFs, which optimizes the pseudolikelihood function for learning.
- **Our Approach.** For SPNs, we try different GNN architectures for defining the node and edge GNNs (e.g., SPN-GAT). By default, we only solve the proxy problem without performing refinement. We systematically compare the results with and without refinement in part 2 of Section 4.5.2.

Evaluation Metrics. On Cora*, Citeseer*, and Pubmed*, we report the percentage of test graphs where all the nodes are correctly classified (i.e., graph-level accuracy). On DBLP and PPI, we report accuracy and micro-F1 based on the percentage of test nodes which are correctly classified (i.e., node-level accuracy). For Cora*, Citeseer*, and Pubmed*, we run each compared method with 10 different seeds to report the mean accuracy and the standard deviation. For DBLP and PPI, we run each method with 5 seeds.

Algorithm	**PPI-1**		**PPI-2**		**PPI-10**		**PPI**	
	Accuracy	Micro-F1	Accuracy	Micro-F1	Accuracy	Micro-F1	Accuracy	Micro-F1
GCN	76.62 ± 0.10	54.55 ± 0.29	77.48 ± 0.12	56.10 ± 0.36	80.43 ± 0.10	62.48 ± 0.27	82.28 ± 0.24	66.52 ± 0.89
GraphSAGE	81.02 ± 0.07	67.30 ± 0.11	84.13 ± 0.04	72.93 ± 0.04	95.34 ± 0.03	92.18 ± 0.05	98.51 ± 0.02	97.51 ± 0.03
GAT	77.49 ± 0.20	60.72 ± 0.25	81.35 ± 0.19	68.55 ± 0.30	96.14 ± 0.15	93.53 ± 0.24	98.85 ± 0.05	98.06 ± 0.08
Graph U-Net	77.17 ± 0.07	55.54 ± 0.33	78.22 ± 0.04	59.12 ± 0.30	83.15 ± 0.04	68.70 ± 0.08	86.29 ± 0.04	75.57 ± 0.18
GCNII	80.99 ± 0.07	65.79 ± 0.25	84.81 ± 0.06	74.54 ± 0.14	97.53 ± 0.01	95.86 ± 0.01	99.39 ± 0.00	98.97 ± 0.00
CRF-linear	65.33 ± 2.77	48.30 ± 0.35	67.20 ± 2.24	49.45 ± 0.97	69.72 ± 0.65	50.17 ± 0.39	69.98 ± 0.30	50.61 ± 0.35
CRF-GCN	76.33 ± 0.21	50.79 ± 0.74	76.27 ± 0.10	49.47 ± 0.63	77.08 ± 0.07	52.36 ± 0.72	77.34 ± 0.07	53.60 ± 0.36
CRF-GraphSAGE	77.43 ± 0.28	54.57 ± 1.07	77.25 ± 0.36	53.48 ± 1.00	77.65 ± 0.38	54.44 ± 1.34	77.21 ± 0.19	54.50 ± 3.09
CRF-GAT	76.50 ± 0.49	52.95 ± 0.40	76.76 ± 0.61	55.01 ± 0.93	74.58 ± 0.92	54.98 ± 1.13	70.42 ± 0.72	53.27 ± 0.42
CRF-GCNII	79.98 ± 0.32	61.22 ± 1.10	81.73 ± 0.33	66.37 ± 0.56	92.11 ± 0.28	87.10 ± 0.40	96.94 ± 0.12	94.95 ± 0.19
GMNN	77.55 ± 0.53	57.20 ± 2.63	81.21 ± 0.87	67.46 ± 2.92	94.67 ± 2.77	90.72 ± 5.28	97.00 ± 2.98	94.69 ± 5.60
SPN-GCN	77.07 ± 0.05	54.15 ± 0.17	78.02 ± 0.05	55.73 ± 0.15	80.59 ± 0.04	61.36 ± 0.11	82.56 ± 0.20	66.70 ± 0.77
SPN-GraphSAGE	**82.11** ± 0.03	**68.56** ± 0.07	85.40 ± 0.05	74.45 ± 0.07	95.28 ± 0.02	91.99 ± 0.04	98.55 ± 0.02	97.56 ± 0.03
SPN-GAT	79.01 ± 0.17	64.02 ± 0.40	83.55 ± 0.12	72.37 ± 0.18	96.68 ± 0.13	94.41 ± 0.21	99.04 ± 0.06	98.38 ± 0.10
SPN-GCNII	82.01 ± 0.03	67.80 ± 0.11	**85.83** ± 0.04	**75.96** ± 0.05	**97.55** ± 0.01	**95.87** ± 0.02	**99.41** ± 0.00	**99.02** ± 0.00

Table 11. Result of SPN on PPI datasets.

Algorithm	Cora*	Citeseer*	Pubmed*	DBLP
GCN	57.26 ± 0.66	46.24 ± 0.61	51.84 ± 0.45	76.60 ± 2.32
GraphSAGE	49.02 ± 2.37	41.32 ± 2.41	48.61 ± 1.28	73.81 ± 0.90
GAT	51.99 ± 3.51	47.94 ± 0.46	50.89 ± 0.52	79.16 ± 1.44
Graph U-Net	56.07 ± 0.57	45.91 ± 1.65	51.77 ± 0.97	75.21 ± 2.68
GCNII	59.15 ± 0.67	46.39 ± 0.92	53.54 ± 0.98	81.79 ± 0.88
CRF-linear	42.78 ± 3.94	40.60 ± 0.81	43.90 ± 2.91	54.26 ± 1.27
CRF-GAT	49.10 ± 3.80	42.89 ± 1.30	47.79 ± 1.33	59.14 ± 4.15
CRF-UNet	53.49 ± 2.47	43.66 ± 2.12	50.02 ± 0.88	57.46 ± 3.07
CRF-GCNII	36.18 ± 5.75	38.27 ± 4.82	41.71 ± 4.79	60.55 ± 2.23
GMNN	54.30 ± 1.15	48.46 ± 1.06	51.70 ± 1.23	76.54 ± 2.93
SPN-GAT	58.78 ± 1.21	**49.02** ± 0.78	52.91 ± 0.54	**84.84** ± 0.73
SPN-UNet	58.03 ± 0.54	46.97 ± 1.06	53.36 ± 0.67	80.11 ± 1.59
SPN-GCNII	**60.47** ± 0.49	48.34 ± 0.50	**54.35** ± 0.64	83.57 ± 1.33

Table 12. Results of SPN on Cora*, Citeseer*, Pubmed*, and DBLP.

Experimental Setup. For GNNs, by default we use the same architectures (e.g., number of neurons, number of layers) as used in the original papers. Adam [59] is used for training. For the edge GNN in Equation (4.4.4), we add a hyperparameter γ to control the annealing temperature of the logit $g(\mathbf{v}_s, \mathbf{v}_t)$ before the softmax function during belief propagation. Empirically, we find that max-product belief propagation works better than the sum-product variant in most cases, so we use the max-product version by default. By default, we do not run refinement when training SPNs. See Section B.6 for details.

4.5.2. Results

1. Comparison with other methods. The main results in the two settings are presented in Table 11 and Table 12, where the results are in %. Compared against different GNN models, our approach achieves consistent improvement (the relative underperformance of SPN-GCN and SPN-SAGE is related to the capacity of the backbone GNNs and is explained in Section 6) by using these GNNs as backbone networks for approximating marginal label distributions on nodes and edges, which demonstrates SPNs are able to model the structured output of node labels by combining with CRFs, and thus achieve better results.

Besides, SPNs also achieve superior results to CRF-GNNs which are trained by directly solving the maximin game in Equation (4.3.3), as well as GMNN which optimizes the pseudolikelihood function. This observation proves the advantage of our proposed proxy optimization problem for learning CRFs.

2. Effect of refinement. By solving the proxy optimization problem in Equation (4.4.5), we can obtain a near-optimal joint label distribution on training graphs, based on which we may optionally refine the distribution with the maximin game in Equation (4.3.3). Next, we study the effect of refinement, and we present the results in Table 14. By only solving

Table 13. Run time of SPN.

Algorithm	DBLP	PPI
GAT	23.15	460.81
CRF (GAT)	500.43	27136.90
SPN(GAT)	46.86	962.92

Table 14. Analysis of refinement in SPN.

Algorithm	Refine	PPI-2	PPI-10	PPI
SPN-	w/o	71.52 ± 0.21	94.41 ± 0.21	98.38 ± 0.10
GAT	with	**71.58** ± 0.20	**94.63** ± 0.20	**98.68** ± 0.09
SPN-	w/o	**73.93** ± 0.08	91.99 ± 0.04	97.56 ± 0.03
GraphSAGE	with	73.68 ± 0.10	**92.49** ± 0.02	**97.77** ± 0.02

the proxy problem, our approach already achieves impressive results, showing that the proxy problem can well approximate the original learning problem. Only on datasets with sufficient labeled data (e.g., PPI-10, PPI), refinement leads to some improvement.

Table 15. Analysis of the proxy problem.

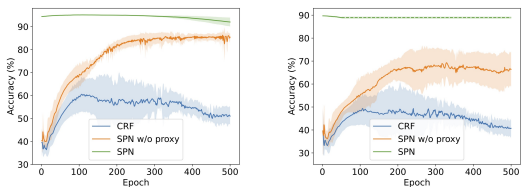
Algorithm	Cora*	Citeseer*	PPI-10
Maximin Game	49.10 ± 3.80	42.89 ± 1.30	54.98 ± 1.13
Pseudolikelihood	54.30 ± 1.15	48.46 ± 1.06	90.72 ± 5.28
Proxy Problem	**58.78** ± 1.21	**49.02** ± 0.78	**95.87** ± 0.02

Table 16. Analysis of SPN variants.

Algorithm	PPI-1	PPI-2	PPI-10
GAT	60.72 ± 0.25	68.55 ± 0.30	93.53 ± 0.24
SPN-GAT node and edge GNNs	**64.02** ± 0.40	**72.37** ± 0.18	94.41 ± 0.21
SPN-GAT a shared GNN	63.72 ± 0.38	70.99 ± 0.25	**95.19** ± 0.15

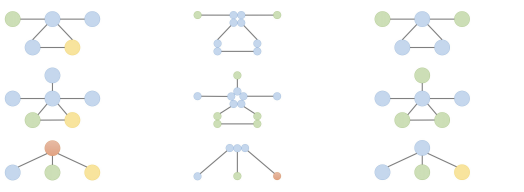
3. Model architecture. SPN uses a node GNN and an edge GNN for computing node and edge marginals independently. In practice, we can also use a shared GNN for both node and edge marginals. We show results of this variant in Table 16, where it also achieves significant improvement over GNNs.

4. Efficiency comparison. We have seen SPNs achieve better classification results than GNNs and CRFs. Next, we further compare their efficiency by showing the run time on DBLP and PPI (in seconds). For PPI, which has 121 labels, we only report the training times on a single label. We use GAT as the backbone network for CRFs and SPNs. GAT and CRF are trained for 1000 epochs to ensure convergence. For the SPN, we train the node GNN and edge GNN for node/edge classification as in Equation (4.4.6) with 1000 epochs. The run times are presented in Table 13. SPNs take twice as long for training than GAT, as a SPN needs to train a node GNN and an edge GNN. Compared with CRFs, SPNs are much more efficient, because the proxy optimization problem in SPNs is much easier to solve.



(a) DBLP training. (b) DBLP validation.

Figure 4. Convergence curves of SPN.



(a) GAT. (b) Edge GNN. (c) SPN-GAT. (same as ground truth)

Figure 5. Case study of SPN.

5. Comparison of learning methods. Next, we investigate different methods for learning SPNs, including directly solving the maximin game, optimizing pseudolikelihood, and solving our proposed proxy problem. We show the results for optimizing SPN-GAT in Table 15. We see solving maximin game yields poor results due to unstable training. Although the pseudolikelihood method performs much better, the result is still unsatisfactory as it is not a good approximation of the true likelihood. By solving our proposed proxy problem, SPN achieves the best result, which proves its effectiveness.

6. Convergence analysis. To better illustrate the advantage of the proxy problem for learning CRFs, we look into the training curves of SPNs, SPNs w/o proxy, and CRFs when optimizing the maximin game in Equation (4.3.3). For SPNs, we optimize the node and edge GNNs on the proxy optimization problem in Equation (4.4.5) before doing refinement with the maximin game, while for SPNs w/o proxy we directly perform refinement with the maximin game without solving the proxy problem. We show the results in Figure 4. CRFs and SPNs w/o proxy suffer from high variance and low accuracy. In contrast, owing to the near-optimal joint distribution found by solving the proxy problem, SPNs get higher accuracy with lower variance even without refinement (see initial results of SPNs at epoch 0). Also, the refinement process quickly converges after only a few epochs, showing good efficiency of SPNs.

7. Case study. To intuitively see how SPN outperform GNNs, we conduct some case studies on Cora*. We use GAT as backbone networks, and show the prediction made by the GAT (the node GNN), the edge GNN, and SPN in Figure 5. In all three cases shown in the figure, GAT (left column) makes inconsistent predictions on linked nodes, as it fails to model the structured output. The edge GNN (middle column) also makes a mistake in the bottom case. Finally, by combining GAT and edge GNN with a CRF, the SPN (right column) is able to predict the correct labels for all nodes.

8. Additional Analysis of GNN Architectures. In this analysis, we study the effect of node/edge GNN architectures on SPNs. We fix one of the GNNs and change the capacity of the other (Figure 6), then evaluate SPN-GAT on **PPI-1-0**, a subset of PPI-1 that only contains its first label. The results show that our model benefit from capacity gain in both node and edge GNNs, highlighting their effective synergy. This also explains the underperformance of SPN-GCN in Table 11, where the edge GCN backbone with only two layers and 16 hidden neurons is incapable of modeling the edge label dependencies and thus drags the performance behind. We also find that the node and edge GNNs need not share the same backbone, and in many cases SPNs with different node and edge GNNs perform superior to those with same backbone (Figure 6). The expressiveness of edge GNNs is crucial to the

performance of SPN. Though we did not optimize the design of our edge GNNs, they have shown to be helpful in boosting the performance once plugged into our approach.

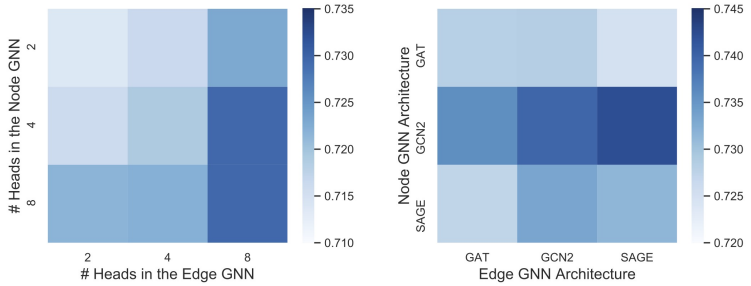


Figure 6. Analysis of GNN architectures in SPN.

9. Node-level Accuracy on Cora*, Citeseer*, and Pubmed*. In the experiment, we report the graph-level accuracy on the Cora*, Citeseer*, and Pubmed* datasets, where SPNs consistently outperform other methods. Besides the graph-level accuracy, we also compute the node-level accuracy on these datasets, and the results are reported in Table 17. We can see that our approach still consistently outperforms other methods in terms of node-level accuracy.

Algorithm	Cora*	Citeseer*	Pubmed*
GCN	79.85 ± 0.24	72.25 ± 0.71	78.05 ± 0.55
GraphSAGE	73.43 ± 1.67	62.48 ± 2.19	73.99 ± 1.26
GAT	79.65 ± 1.25	74.15 ± 0.12	78.62 ± 0.52
Graph U-Net	78.72 ± 0.63	71.36 ± 1.37	77.93 ± 0.60
GCNII	82.84 ± 0.37	72.61 ± 0.49	79.47 ± 0.55
CRF-linear	68.47 ± 2.13	65.88 ± 0.85	65.93 ± 2.18
CRF-GAT	77.75 ± 1.24	69.13 ± 1.10	75.96 ± 1.06
CRF-UNet	78.32 ± 1.51	70.78 ± 1.15	77.91 ± 0.56
CRF-GCNII	35.98 ± 7.40	33.73 ± 5.87	60.55 ± 4.17
GMNN	79.90 ± 0.93	72.18 ± 0.48	78.00 ± 1.04
SPN-GAT	83.13 ± 0.48	**74.50** ± 0.36	79.23 ± 0.33
SPN-UNet	81.11 ± 0.55	72.28 ± 0.94	78.70 ± 0.37
SPN-GCNII	**83.54** ± 0.27	74.04 ± 0.29	**79.95** ± 0.38

Table 17. Node-level accuracy of SPN on Cora*, Citeseer*, Pubmed*.

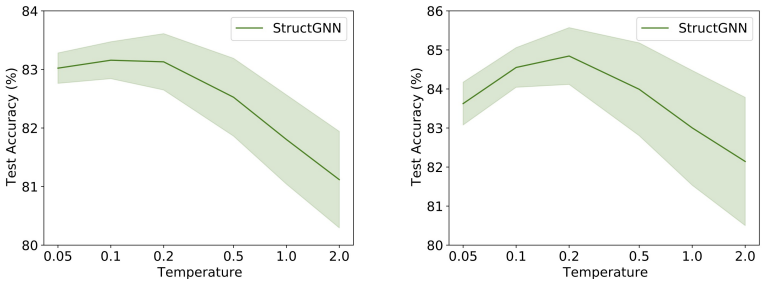
10. Comparison of Sum-product and Max-product Belief Propagation. As explained in Section 4.4.2, the sum-product belief propagation algorithm is more applicable to the case of node-level accuracy, as it aims at inferring the marginal label distribution on each node. Nevertheless, in practice we find that the max-product algorithm usually achieves

better empirical node-level accuracy. For example, the results on the PPI-10 dataset are presented in Table 18. Because of the better empirical results, we choose to use max-product belief propagation by default.

Algorithm	Micro-F1
Sum-product BP	94.50 ± 0.16
Max-product BP	94.65 ± 0.13

Table 18. Analysis of belief propagation in SPN.

11. Hyperparameter Analysis. Finally, We present analysis of the hyperparameter γ (i.e., edge temperature) in Figure 7. We can see that $[0.1, 0.2]$ is a good default range for γ.



(a) Edge temperature on Cora. (b) Edge temperature on DBLP.

Figure 7. Analysis of hyperparameters in SPN.

4.6. Conclusion

This paper studied inductive node classification, and we proposed SPN to combine GNNs and CRFs. Inspired by the connection of joint and marginal distributions defined by Markov networks, we designed a proxy problem for efficient model learning. In the future, we plan to explore more advanced GNNs to model the pseudomarginals on edges, which are key to improving node classification results in SPNs. In addition, SPNs model joint dependency of node labels by defining potential functions on nodes and edges, and we also plan to further explore high-order local structures, e.g., triangles.

Chapter 5

pLogicNet: Probabilistic Logic Neural Nets

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

5.1. Introduction

Many real-world entities are interconnected with each other through various types of relationships, forming massive relational data. Naturally, such relational data can be characterized by a set of (h, r, t) triplets, meaning that entity h has relation r with entity t. To store the triplets, many knowledge graphs have been constructed such as Freebase [44] and WordNet [83]. These graphs have been proven useful in many tasks, such as question answering [161], relation extraction [102] and recommender systems [10]. However, one big

challenge of knowledge graphs is that their coverage is limited. Therefore, one fundamental problem is how to predict the missing links based on the existing triplets.

One type of methods for reasoning on knowledge graphs are the symbolic logic rule-based approaches [40, 51, 106, 127, 149]. These rules can be either handcrafted by domain experts [129] or mined from knowledge graphs themselves [32]. Traditional methods such as expert systems [40, 51] use hard logic rules for prediction. For example, given a logic rule $\forall x,y, \mathit{Husband}(x,y) \Rightarrow \mathit{Wife}(y,x)$ and a fact that A is the husband of B, we can derive that B is the wife of A. However, in many cases logic rules can be imperfect or even contradictory, and hence effectively modeling the uncertainty of logic rules is very critical. A more principled method for using logic rules is the Markov Logic Network (MLN) [106, 119], which combines first-order logic and probabilistic graphical models. MLNs learn the weights of logic rules in a probabilistic framework and thus soundly handle the uncertainty. Such methods have been proven effective for reasoning on knowledge graphs. However, the inference process in MLNs is difficult and inefficient due to the complicated graph structure among triplets. Moreover, the results can be unsatisfactory as many missing triplets cannot be inferred by any rules.

Another type of methods for reasoning on knowledge graphs are the recent knowledge graph embedding based methods (e.g., TransE [8], DistMult [156] and ComplEx [135]). These methods learn useful embeddings of entities and relations by projecting existing triplets into low-dimensional spaces. These embeddings preserve the semantic meanings of entities and relations, and can effectively predict the missing triplets. In addition, they can be efficiently trained with stochastic gradient descent. However, one limitation is that they do not leverage logic rules, which compactly encode domain knowledge and are useful in many applications.

We are seeking an approach that combines the advantages of both worlds, one which is able to exploit first-order logic rules while handling their uncertainty, infer missing triplets effectively, and can be trained in an efficient way. We propose such an approach called the probabilistic Logic Neural Networks (pLogicNet). A pLogicNet defines the joint distribution of a collection of triplets with a Markov Logic Network [106], which associates each logic rule with a weight and can be effectively trained with the variational EM algorithm [89]. In the variational E-step, we infer the plausibility of the unobserved triplets (i.e., hidden variables) with amortized mean-field inference [36, 60, 95], in which the variational distribution is parameterized as a knowledge graph embedding model. In the M-step, we update the weights of logic rules by optimizing the pseudolikelihood [6], which is defined on both the observed triplets and those inferred by the knowledge graph embedding model. The framework can be efficiently trained by stochastic gradient descent. Experiments on four benchmark knowledge graphs prove the effectiveness of pLogicNet over many competitive baselines.

5.2. Related Work

First-order logic rules can compactly encode domain knowledge and have been extensively explored for reasoning. Early methods such as expert systems [40, 51] use hard logic rules for reasoning. However, logic rules can be imperfect or even contradictory. Later studies try to model the uncertainty of logic rules by using Horn clauses [20, 56, 98, 149] or database query languages [99, 127]. A more principled method is the Markov logic network [106, 119], which combines first-order logic with probabilistic graphical models. Despite the effectiveness in a variety of tasks, inference in MLNs remains difficult and inefficient due to the complicated connections between triplets. Moreover, for predicting missing triplets on knowledge graphs, the performance can be limited as many triplets cannot be discovered by any rules. In contrast to them, pLogicNet uses knowledge graph embedding models for inference, which is much more effective by learning useful entity and relation embeddings.

Another category of approach for knowledge graph reasoning is the knowledge graph embedding method [8, 26, 92, 121, 135, 146, 156], which aims at learning effective embeddings of entities and relations. Generally, these methods design different scoring functions to model different relation patterns for reasoning. For example, TransE [8] defines each relation as a translation vector, which can effectively model the composition and inverse relation patterns. DistMult [156] models the symmetric relation with a bilinear scoring function. ComplEx [135] models the asymmetric relations by using a bilinear scoring function in complex space. RotatE [121] further models multiple relation patterns by defining each relation as a rotation in complex spaces. Despite the effectiveness and efficiency, these methods are not able to leverage logic rules, which are beneficial in many tasks. Recently, there are a few studies on combining logic rules and knowledge graph embedding [27, 45]. However, they cannot effectively handle the uncertainty of logic rules. Compared with them, pLogicNet is able to use logic rules and also handle their uncertainty in a more principled way through Markov logic networks.

Some recent work also studies using reinforcement learning for reasoning on knowledge graphs [23, 76, 117, 153], where an agent is trained to search for reasoning paths. However, the performance of these methods is not so competitive. Our pLogicNets are easier to train and also more effective.

Lastly, there are also some recent studies trying to combine statistical relational learning and graph neural networks for semi-supervised node classification [103], or using Markov networks for visual dialog reasoning [100, 168]. Our work shares similar idea with these studies, but we focus on a different problem, i.e., reasoning with first-order logic on knowledge graphs. There is also a concurrent work using graph neural networks for logic reasoning [165].

Compared to this study which emphasizes more on the inference problem, our work focuses on both the inference and the learning problems.

5.3. Preliminary

5.3.1. Problem Definition

A knowledge graph is a collection of relational facts, each of which is represented as a triplet (h,r,t). Due to the high cost of knowledge graph construction, the coverage of knowledge graphs is usually limited. Therefore, a critical problem on knowledge graphs is to predict the missing facts.

Formally, given a knowledge graph (E,R,O), where E is a set of entities, R is a set of relations, and O is a set of observed (h,r,t) triplets, the goal is to infer the missing triplets by reasoning with the observed triplets. Following existing studies [91], the problem can be reformulated in a probabilistic way. Each triplet (h,r,t) is associated with a binary indicator variable $\mathbf{v}_{(h,r,t)}$. $\mathbf{v}_{(h,r,t)} = 1$ means (h,r,t) is true, and $\mathbf{v}_{(h,r,t)} = 0$ otherwise. Given some true facts $\mathbf{v}_O = \{\mathbf{v}_{(h,r,t)} = 1\}_{(h,r,t)\in O}$, we aim to predict the labels of the remaining hidden triplets H, i.e., $\mathbf{v}_H = \{\mathbf{v}_{(h,r,t)}\}_{(h,r,t)\in H}$. We will discuss how to generate the hidden triplets H later in Section 5.4.4.

This problem has been extensively studied in both traditional logic rule-based methods and recent knowledge graph embedding methods. For logic rule-based methods, we mainly focus on one representative approach, the Markov logic network [106]. Essentially, both types of methods aim to model the joint distribution of the observed and hidden triplets $p(\mathbf{v}_O,\mathbf{v}_H)$. Next, we briefly introduce the Markov logic network (MLN) [106] and the knowledge graph embedding methods [8, 121, 156].

5.3.2. Markov Logic Network

In the MLN, a Markov network is designed to define the joint distribution of the observed and the hidden triplets, where the potential function is defined by the first-order logic. Some common logic rules to encode domain knowledge include: (1) **Composition Rules**. A relation r_k is a composition of r_i and r_j means that for any three entities x, y, z, if x has relation r_i with y, and y has relation r_j with z, then x has relation r_k with z. Formally, we have $\forall x, y, z \in E, \mathbf{v}_{(x,r_i,y)} \wedge \mathbf{v}_{(y,r_j,z)} \Rightarrow \mathbf{v}_{(x,r_k,z)}$. (2) **Inverse Rules**. A relation r_j is an inverse of r_i indicates that for two entities x, y, if x has relation r_i with y, then y has relation r_j with x. We can represent the rule as $\forall x, y \in E, \mathbf{v}_{(x,r_i,y)} \Rightarrow \mathbf{v}_{(y,r_j,x)}$. (3) **Symmetric Rules**. A relation r is symmetric means that for any entity pair x, y, if x has relation r with y, then y also has relation r with x. Formally, we have $\forall x, y \in E, \mathbf{v}_{(x,r,y)} \Rightarrow \mathbf{v}_{(y,r,x)}$.

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

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

$$p(\mathbf{v}_O,\mathbf{v}_H) = \frac{1}{Z}\exp\left(\sum_{l\in L} w_l \sum_{g\in G_l} \mathbb{I}\{g \text{ is true}\}\right) = \frac{1}{Z}\exp\left(\sum_{l\in L} w_l n_l(\mathbf{v}_O,\mathbf{v}_H)\right), \tag{5.3.1}$$

where n_l is the number of true groundings of rule l based on the values of $\mathbf{v}_O$ and $\mathbf{v}_H$.

With such a formulation, predicting the missing triplets essentially becomes inferring the posterior distribution $p(\mathbf{v}_H|\mathbf{v}_O)$. Exact inference is usually infeasible due to the complicated graph structures, and hence approximation inference is often used such as MCMC [41] and loopy belief propagation [85].

5.3.3. Knowledge Graph Embedding

Different from the logic rule-based approaches, the knowledge graph embedding methods learn embeddings of entities and relations with the observed facts $\mathbf{v}_O$, and then predict the missing facts with the learned entity and relation embeddings. Formally, each entity $e \in E$ and relation $r \in R$ is associated with an embedding $\mathbf{x}_e$ and $\mathbf{x}_r$. Then the joint distribution of all the triplets is defined as:

$$p(\mathbf{v}_O, \mathbf{v}_H) = \prod_{(h,r,t)\in O\cup H} \text{Ber}(\mathbf{v}_{(h,r,t)}|f(\mathbf{x}_h, \mathbf{x}_r, \mathbf{x}_t)), \tag{5.3.2}$$

where Ber stands for the Bernoulli distribution, $f(\mathbf{x}_h, \mathbf{x}_r, \mathbf{x}_t)$ computes the probability that the triplet (h,r,t) is true, with $f(\cdot,\cdot,\cdot)$ being a scoring function on the entity and relation embeddings. For example in TransE, the score function f can be formulated as $\sigma(\gamma - ||\mathbf{x}_h + \mathbf{x}_r - \mathbf{x}_t||)$ according to [121], where σ is the sigmoid function and γ is a fixed bias. To learn the entity and relation embeddings, these methods typically treat observed triplets as positive examples and the hidden triplets as negative ones. In other words, these methods seek to maximize $\log p(\mathbf{v}_O = 1, \mathbf{v}_H = 0)$. The whole framework can be efficiently optimized with the stochastic gradient descent algorithm.

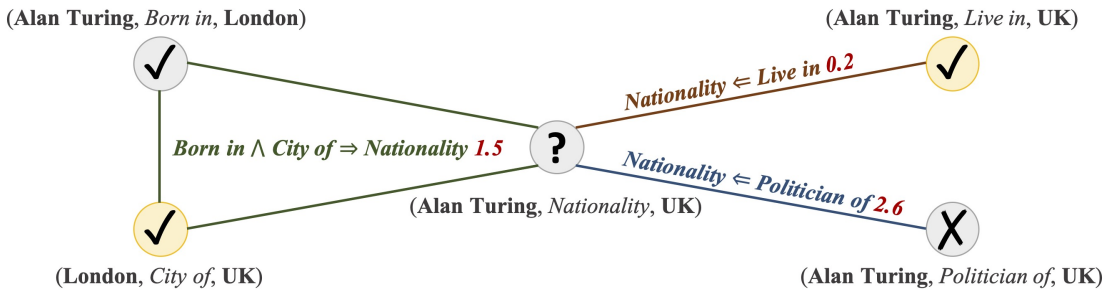


Figure 8. Overview of pLogicNet.

5.4. Model

In this section, we introduce our proposed approach pLogicNet for knowledge graph reasoning, which combines the logic rule-based methods and the knowledge graph embedding methods. To leverage the domain knowledge provided by first-order logic rules, pLogicNet formulates the joint distribution of all triplets with a Markov logic network [106], which is trained with the variational EM algorithm [89], alternating between a variational E-step and an M-step. In the varational E-step, we use a knowledge graph embedding model to infer the missing triplets, during which the knowledge preserved by the logic rules can be effectively distilled into the learned embeddings. In the M-step, the weights of the logic rules are updated based on both the observed triplets and those inferred by the knowledge graph embedding model. In this way, the embedding model provides extra supervision for weight learning.

An overview of pLogicNet is given in Figure 8. Each possible triplet is associated with a binary indicator (circles), indicating whether it is true (✓) or not (✗). The observed (yellow circles) and hidden (grey circles) indicators are connected by a set of logic rules, with each rule having a weight (red number). For the center triplet, the KGE model predicts its indicator through embeddings, while the logic rules consider the Markov blanket of the triplet (all connected triplets). If any indicator in the Markov blanket is hidden, we simply fill it with the prediction from the KGE model. In the **E-step**, we use the logic rules to predict the center indicator, and treat it as extra training data for the KGE model. In the **M-step**, we annotate all hidden indicators with the KGE model, and then update the weights of rules.

5.4.1. Variational EM

Given a set of first-order logic rules $L = \{l_i\}_{i=1}^{|L|}$, our approach uses a Markov logic network [106] as in Equation (5.3.1) to model the joint distribution:

$$p_w(\mathbf{v}_O, \mathbf{v}_H) = \frac{1}{Z} \exp\left(\sum_l w_l n_l(\mathbf{v}_O, \mathbf{v}_H)\right), \tag{5.4.1}$$

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

$$\log p_w(\mathbf{v}_O) \geq \mathcal{L}(q_\theta, p_w) = \mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_w(\mathbf{v}_O,\mathbf{v}_H) - \log q_\theta(\mathbf{v}_H)], \tag{5.4.2}$$

where $q_\theta(\mathbf{v}_H)$ is a variational distribution of the hidden variables $\mathbf{v}_H$. The equation holds when $q_\theta(\mathbf{v}_H)$ equals to the true posterior distribution $p_w(\mathbf{v}_H|\mathbf{v}_O)$. Such a lower bound can be effectively optimized with the variational EM algorithm [89], which consists of a variational E-step and an M-step. In the variational E-step, which is known as the inference procedure, we fix p_w and update q_θ to minimize the KL divergence between $q_\theta(\mathbf{v}_H)$ and $p_w(\mathbf{v}_H|\mathbf{v}_O)$. In the M-step, which is known as the learning procedure, we fix q_θ and update p_w to maximize the log-likelihood function of all the triplets, i.e., $\mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_w(\mathbf{v}_O,\mathbf{v}_H)]$. Next, we introduce the details of both steps.

5.4.2. E-step: Inference Procedure

For inference, we aim to infer the posterior distribution of the hidden variables, i.e., $p_w(\mathbf{v}_H|\mathbf{v}_O)$. As exact inference is intractable, we approximate the true posterior distribution with a mean-field [95] variational distribution $q_\theta(\mathbf{v}_H)$, in which each $\mathbf{v}_{(h,r,t)}$ is inferred independently for $(h,r,t) \in H$. To further improve inference, we use amortized inference [36, 60], and parameterize $q_\theta(\mathbf{v}_{(h,r,t)})$ with a knowledge graph embedding model. Formally, $q_\theta(\mathbf{v}_H)$ is formulated as below:

$$q_\theta(\mathbf{v}_H) = \prod_{(h,r,t)\in H} q_\theta(\mathbf{v}_{(h,r,t)}) = \prod_{(h,r,t)\in H} \mathrm{Ber}(\mathbf{v}_{(h,r,t)}|f(\mathbf{x}_h,\mathbf{x}_r,\mathbf{x}_t)), \tag{5.4.3}$$

where Ber stands for the Bernoulli distribution, and $f(\cdot,\cdot,\cdot)$ is a scoring function defined on triplets as introduced in Section 5.3.3. By minimizing the KL divergence between the variational distribution $q_\theta(\mathbf{v}_H)$ and the true posterior $p_w(\mathbf{v}_H|\mathbf{v}_O)$, the optimal $q_\theta(\mathbf{v}_H)$ is given by the fixed-point condition for all $(h,r,t) \in H$:

$$\log q_\theta(\mathbf{v}_{(h,r,t)}) = \mathbb{E}_{q_\theta(\mathbf{v}_{\mathrm{MB}(h,r,t)})}[\log p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_{\mathrm{MB}(h,r,t)})] + \mathrm{const} \tag{5.4.4}$$

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

With Equation (5.4.4), our goal becomes finding a distribution q_θ that satisfies the condition. However, Equation (5.4.4) involves the expectation with respect to $q_\theta(\mathbf{v}_{\mathrm{MB}(h,r,t)})$.

To simplify the condition, we follow [50] and estimate the expectation with a sample $\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)} = \{\hat{\mathbf{v}}_{(h',r',t')}\}_{(h',r',t')\in\mathrm{MB}(h,r,t)}$. Specifically, for each $(h',r',t') \in \mathrm{MB}(h,r,t)$, if it is observed, we set $\hat{\mathbf{v}}_{(h',r',t')} = 1$, and otherwise $\hat{\mathbf{v}}_{(h',r',t')} \sim q_\theta(\mathbf{v}_{(h',r',t')})$. In this way, the right side of Equation (5.4.4) is approximated as $\log p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$, and thus the optimality condition can be further simplified as $q_\theta(\mathbf{v}_{(h,r,t)}) \approx p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$.

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

To learn the optimal q_θ, we use a method similar to [109]. We start by computing $p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$ with the current q_θ. Then, we fix the value as target, and update q_θ to minimize the reverse KL divergence of $q_\theta(\mathbf{v}_{(h,r,t)})$ and the target $p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$, leading to the following objective:

$$O_{\theta,U} = \sum_{(h,r,t)\in H} \mathbb{E}_{p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})}[\log q_\theta(\mathbf{v}_{(h,r,t)})]. \tag{5.4.5}$$

To optimize this objective, we first compute $p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$ for each hidden triplet (h,r,t). If $p_w(\mathbf{v}_{(h,r,t)} = 1|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)}) \geq \tau_{\mathrm{triplet}}$ with τ_{triplet} being a hyperparameter, then we treat (h,r,t) as a positive example and train the knowledge graph embedding model to maximize the log-likelihood $\log q_\theta(\mathbf{v}_{(h,r,t)} = 1)$. Otherwise the triplet is treated as a negative example. In this way, the knowledge captured by logic rules can be effectively distilled into the knowledge graph embedding model.

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

$$O_{\theta,L} = \sum_{(h,r,t)\in O} \log q_\theta(\mathbf{v}_{(h,r,t)} = 1). \tag{5.4.6}$$

By adding Equation (5.4.5) and Equation (5.4.6), we get the overall objective for q_θ, i.e., $O_\theta = O_{\theta,U} + O_{\theta,L}$.

5.4.3. M-step: Learning Procedure

In the learning procedure, we will fix q_θ, and update the weights of logic rules w by maximizing the log-likelihood function, i.e., $\mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_w(\mathbf{v}_O,\mathbf{v}_H)]$. However, directly optimizing the log-likelihood function can be difficult, as we need to deal with the partition function, i.e.,

Z in Equation (5.4.1). Therefore, we follow existing studies [62, 106] and instead optimize the pseudolikelihood function [6]:

$$\ell_{PL}(w) \triangleq \mathbb{E}_{q_\theta(\mathbf{v}_H)}[\sum_{h,r,t} \log p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_{O\cup H\setminus(h,r,t)})] = \mathbb{E}_{q_\theta(\mathbf{v}_H)}[\sum_{h,r,t} \log p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_{\mathrm{MB}(h,r,t)})],$$

where the second equation is based on the independence property of the MLN in Equation (5.4.1).

We optimize w through the gradient descent algorithm. For each expected conditional distribution $\mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_{\mathrm{MB}(h,r,t)})]$, suppose $\mathbf{v}_{(h,r,t)}$ connects with $\mathbf{v}_{\mathrm{MB}(h,r,t)}$ through a set of rules. For each of such rules l, the derivative with respect to w_l is:

$$\nabla_{w_l}\mathbb{E}_{q_\theta(\mathbf{v}_H)}[\log p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_{\mathrm{MB}(h,r,t)})] \simeq y_{(h,r,t)} - p_w(\mathbf{v}_{(h,r,t)} = 1|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)}) \tag{5.4.7}$$

where $y_{(h,r,t)} = 1$ if (h,r,t) is an observed triplet and $y_{(h,r,t)} = q_\theta(\mathbf{v}_{(h,r,t)} = 1)$ if (h,r,t) is hidden. $\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)} = \{\hat{\mathbf{v}}_{(h',r',t')}\}_{(h',r',t')\in\mathrm{MB}(h,r,t)}$ is a sample from q_θ. For each $(h',r',t') \in \mathrm{MB}(h,r,t)$, $\hat{\mathbf{v}}_{(h',r',t')} = 1$ if (h',r',t') is observed, and otherwise $\hat{\mathbf{v}}_{(h',r',t')} \sim q_\theta(\mathbf{v}_{(h',r',t')})$.

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

5.4.4. Optimization and Prediction

During training, we iteratively perform the E-step and the M-step until convergence. Note that there are a huge number of possible hidden triplets (i.e., $|E| \times |R| \times |E| - |O|$), and handling all of them is impractical for optimization. Therefore, we only include a small number of triplets in the hidden set H. Specifically, an unobserved triplet (h,r,t) is added to H if we can find a grounding $[premise] \Rightarrow [hypothesis]$, where the hypothesis is (h,r,t) and the premise only contains triplets in the observed set O. In practice, we can construct H with brute-force search as in [45].

After training, according to the fixed-point condition given in Equation (5.4.4), the posterior distribution $p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_O)$ for $(h, r, t) \in H$ can be characterized by either $q_\theta(\mathbf{v}_{(h,r,t)})$ or $p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$ with $\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)} \sim q_\theta(\mathbf{v}_{\mathrm{MB}(h,r,t)})$. Although we try to encourage the consensus of p_w and q_θ during training, they may still give different predictions as different information is used. Therefore, we use both of them for prediction, and we approximate the true posterior distribution $p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_O)$ as:

$$p_w(\mathbf{v}_{(h,r,t)}|\mathbf{v}_O) \propto \left\{q_\theta(\mathbf{v}_{(h,r,t)}) + \lambda p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})\right\}, \tag{5.4.8}$$

where λ is a hyperparameter for combining $q_\theta(\mathbf{v}_{(h,r,t)})$ and $p_w(\mathbf{v}_{(h,r,t)}|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$. In practice, we also expect to infer the plausibility of the triplets outside H. For each of such triplets

(h,r,t), we can still compute $q_\theta(\mathbf{v}_{(h,r,t)})$ through the learned embeddings, but we cannot make predictions with the logic rules, so we simply replace $p_w(\mathbf{v}_{(h,r,t)} = 1|\hat{\mathbf{v}}_{\mathrm{MB}(h,r,t)})$ with 0.5 in Equation (5.4.8).

5.5. Experiment

5.5.1. Settings

Datasets. In experiments, we evaluate the pLogicNet on four benchmark datasets. The FB15k [8] and FB15k-237 [132] datasets are constructed from Freebase [7]. WN18 [8] and WN18RR [26] are constructed from WordNet [83]. The detailed statistics of the datasets are summarized in Section C.1.

Evaluation Metrics. We compare different methods on the task of knowledge graph reasoning. For each test triplet, we mask the head or the tail entity, and let each compared method predict the masked entity. Following existing studies [8, 156], we use the filtered setting during evaluation. The Mean Rank (MR), Mean Reciprocal Rank (MRR) and Hit@K (H@K) are treated as the evaluation metrics.

Compared Algorithms. We compare with both the knowledge graph embedding methods and rule-based methods. For the knowledge graph embedding methods, we choose five representative methods to compare with, including TransE [8], DistMult [156], HolE [92], ComplEx [135] and ConvE [26]. For the rule-based methods, we compare with the Markov logic network (MLN) [106] and the Bayesian logic programming (BLP) method [24], which model logic rules with Markov networks and Bayesian networks respectively. Besides, we also compare with RUGE [45] and NNE-AER [27], which are hybrid methods that combine knowledge graph embedding and logic rules. As only the results on the FB15k dataset are reported in the RUGE paper, we only compare with RUGE on that dataset. For our approach, we consider two variants, where **pLogicNet** uses only q_θ to infer the plausibility of unobserved triplets during evaluation, while **pLogicNet*** uses both q_θ and p_w through Equation (5.4.8).

Experimental Setup of pLogicNet. To generate the candidate rules in the pLogicNet, we search for all the possible composition rules, inverse rules, symmetric rules and subrelations rules from the observed triplets, which is similar to [32, 45]. Then, we compute the empirical precision of each rule, i.e. $p_l = \frac{|S_l \cap O|}{|S_l|}$, where S_l is the set of triplets extracted by the rule l and O is the set of the observed triplets. We only keep rules whose empirical precision is larger than a threshold τ_{rule}. TransE [8] is used as the default knowledge graph embedding

model to parameterize q_θ. We update the weights of logic rules with gradient descent. The detailed hyperparameters settings are available in Table 19.

Dataset	# Entities	# Relations	# Training	# Validation	# Test
FB15k	14,951	1,345	483,142	50,000	59,071
WN18	40,943	18	141,442	5,000	5,000
FB15k-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134

Table 19. Statistics of datasets used in pLogicNet.

5.5.2. Results

Category	Algorithm	FB15k					WN18				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
KGE	TransE [8]	40	0.730	64.5	79.3	86.4	272	0.772	70.1	80.8	92.0
	DistMult [53]	42	0.798	-	-	89.3	655	0.797	-	-	94.6
	HolE [92]	-	0.524	40.2	61.3	73.9	-	0.938	93.0	94.5	94.9
	ComplEx [135]	-	0.692	59.9	75.9	84.0	-	0.941	93.6	94.5	94.7
	ConvE [26]	51	0.657	55.8	72.3	83.1	374	0.943	93.5	94.6	95.6
Rule-based	BLP [24]	415	0.242	15.1	26.9	42.4	736	0.643	53.7	71.7	83.0
	MLN [106]	352	0.321	21.0	37.0	55.0	717	0.657	55.4	73.1	83.9
Hybrid	RUGE [45]	-	0.768	70.3	81.5	86.5	-	-	-	-	-
	NNE-AER [27]	-	0.803	76.1	83.1	87.4	-	0.943	**94.0**	94.5	94.8
Ours	pLogicNet	**33**	0.792	71.4	85.7	90.1	255	0.832	71.6	94.4	95.7
	pLogicNet*	**33**	**0.844**	**81.2**	**86.2**	**90.2**	**254**	**0.945**	93.9	**94.7**	**95.8**

Table 20. Results of pLogicNet on the FB15k and WN18 datasets.

Category	Algorithm	FB15k-237					WN18RR				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
KGE	TransE [8]	181	0.326	22.9	36.3	52.1	3410	0.223	1.3	40.1	53.1
	DistMult [53]	254	0.241	15.5	26.3	41.9	5110	0.43	39	44	49
	ComplEx [135]	339	0.247	15.8	27.5	42.8	5261	**0.44**	**41**	**46**	51
	ConvE [26]	244	0.325	**23.7**	35.6	50.1	4187	0.43	40	44	52
Rule-based	BLP [24]	1985	0.092	6.2	9.8	15.0	12051	0.254	18.7	31.3	35.8
	MLN [106]	1980	0.098	6.7	10.3	16.0	11549	0.259	19.1	32.2	36.1
Ours	pLogicNet	**173**	0.330	23.1	**36.9**	**52.8**	3436	0.230	1.5	41.1	53.1
	pLogicNet*	**173**	**0.332**	**23.7**	36.7	52.4	**3408**	**0.441**	39.8	44.6	**53.7**

Table 21. Results of pLogicNet on the FB15k-237 and WN18RR datasets.

1. Comparing pLogicNet with Other Methods. The main results on the four datasets are presented in Table 20 and Table 21, where H@K is in %. We can see that the pLogicNet significantly outperforms the rule-based methods, as pLogicNet uses a knowledge graph embedding model to improve inference. pLogicNet also outperforms all the knowledge graph

embedding methods in most cases, where the improvement comes from the capability of exploring the knowledge captured by the logic rules. Moreover, our approach is superior to both hybrid methods (RUGE and NNE-AER) under most metrics, as it handles the uncertainty of logic rules in a more principled way.

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

2. Analysis of Different Rule Patterns. In pLogicNet, four types of rule patterns are used. Next, we systematically study the effect of each rule pattern. We take the FB15k and FB15k-237 datasets as examples, and report the results obtained with each single rule pattern in Table 22. On both datasets, most rule patterns can lead to significant improvement compared to the model without logic rules. Moreover, the effects of different rule patterns are quite different across datasets. On FB15k, the inverse and symmetric rules are more important, whereas on FB15k-237, the composition and subrelation rules are more effective.

Rule Pattern	FB15k					FB15k-237				
	MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
Without	40	0.730	64.7	79.4	86.4	181	0.326	22.9	36.3	52.1
Composition	40	0.752	69.3	78.7	86.0	173	**0.335**	**24.1**	**37.1**	**52.5**
Inverse	**39**	**0.813**	**77.7**	**83.1**	**88.1**	175	0.332	23.8	36.7	52.4
Symmetric	40	0.793	75.0	81.7	87.1	175	0.333	23.8	36.8	52.4
Subrelation	40	0.761	70.2	79.8	86.6	**172**	0.334	23.9	36.8	**52.5**

Table 22. Analysis of different rule patterns in pLogicNet.

3. Inference with Different Knowledge Graph Embedding Methods. In this part, we compare the performance of pLogicNet with different knowledge graph embedding methods for inference. We use TransE as the default model and compare with two other widely-used knowledge graph embedding methods, DistMult [156] and ComplEx [135]. The results on the FB15k and WN18RR datasets are presented in Table 23. Comparing with the results in Table 20 and Table 21, we see that pLogicNet improves the performance of all the three methods by using logic rules. Moreover, the pLogicNet achieves very robust performance with any of the three methods for inference.

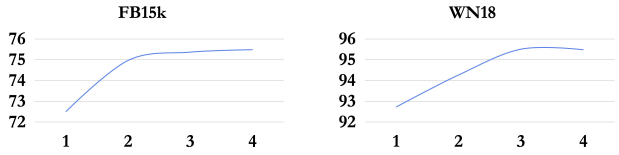
4. Effect of Knowledge Graph Embedding on Logic Rules. In the M-step of pLogicNet, we use the learned embeddings to annotate the hidden triplets, and further update the weights of logic rules. Next, we analyze the effect of knowledge graph embeddings on logic rules. Recall that in the E-step, the logic rules are used to annotate the hidden triplets

KGE Method	Algorithm	FB15k					WN18RR				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
TransE	pLogicNet	33	0.792	71.4	85.7	90.1	3436	0.230	1.5	41.1	53.1
	pLogicNet*	33	0.844	81.2	86.2	90.2	3408	0.441	39.8	44.6	53.7
DistMult	pLogicNet	40	0.791	73.1	83.2	89.5	4902	0.442	39.8	45.5	53.5
	pLogicNet*	39	0.815	76.8	84.6	89.8	4894	0.443	39.9	45.5	53.6
ComplEx	pLogicNet	39	0.776	70.6	81.7	88.5	5266	0.471	43.0	49.2	55.7
	pLogicNet*	45	0.788	73.5	82.1	88.5	5233	0.475	43.5	49.2	55.7

Table 23. Comparison of using different knowledge graph embedding methods in pLogicNet.

Iteration	FB15k		WN18	
	# Triplets	Precision	# Triplets	Precision
1	64,929	79.21%	11,146	80.99%
2	74,717	79.31%	11,430	82.06%
3	76,268	79.10%	11,432	82.09%

Table 24. Effect of KGE on logic rules in pLogicNet.



(a) Cora (b) Citeseer

Figure 9. Convergence analysis of pLogicNet.

through Equation (5.4.5), and thus collect extra positive training data for embedding learning. To evaluate the performance of logic rules, in each iteration we report the number of positive triplets discovered by logic rules, as well as the precision of the triplets in Table 24. We see that as training proceeds, the logic rules can find more triplets with stable precision. This observation proves that the knowledge graph embedding model can indeed provide effective supervision for learning the weights of logic rules.

5. Convergence Analysis. Finally, we present the convergence curves of pLogicNet* on the FB15k and WN18 datasets in Figure 9. The horizontal axis represents the iteration, and the vertical axis shows the value of Hit@1 (in %). We see that on both datasets, our approach takes only 2-3 iterations to converge, which is very efficient.

5.6. Conclusion

This paper studies knowledge graph reasoning, and an approach called the pLogicNet is proposed to integrate existing rule-based methods and knowledge graph embedding methods.

pLogicNet models the distribution of all the possible triplets with a Markov logic network, which is efficiently optimized with the variational EM algorithm. In the E-step, a knowledge graph embedding model is used to infer the hidden triplets, whereas in the M-step, the weights of rules are updated based on the observed and inferred triplets. Experimental results prove the effectiveness of pLogicNet. In the future, we plan to explore more advanced models for inference, such as relational GCN [114] and RotatE [121].

Chapter 6

RNNLogic: Learning Logic Rules

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, this paper proposes a probabilistic model called RNNLogic. RNNLogic treats logic rules as a latent variable, and simultaneously trains a rule generator as well as a reasoning predictor with logic rules. We develop an EM-based algorithm for optimization. In each iteration, the reasoning predictor is first updated to explore some generated logic rules for reasoning. Then in the E-step, we select a set of high-quality rules from all generated rules with both the rule generator and reasoning predictor via posterior inference; and in the M-step, the rule generator is updated with the rules selected in the E-step. Experiments on four datasets prove the effectiveness of RNNLogic.

6.1. Introduction

Knowledge graphs are collections of real-world facts, which are useful in various applications. Each fact is typically specified as a triplet $(h, \mathtt{r}, t)$ or equivalently $\mathtt{r}(h,t)$, meaning entity h has relation $\mathtt{r}$ 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 studies learning logic rules for reasoning on knowledge graphs. For example, one may extract a rule $\forall X,Y,Z$ $\mathtt{hobby}(X,Y) \leftarrow \mathtt{friend}(X,Z) \wedge \mathtt{hobby}(Z,Y)$, meaning that if Z is a friend of X and Z has hobby Y, then Y is also likely the hobby of X. Then the rule can be applied to infer new hobbies of people. Such logic rules are able to improve

interpretability and precision of reasoning [101, 166]. Moreover, logic rules can also be reused and generalized to other domains and data [130]. However, due to the large search space, 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. Most traditional methods such as path ranking [71] and Markov logic networks [106] enumerate relational paths on graphs as candidate logic rules, and then learn a weight for each rule as an assessment of rule qualities. There are also some recent methods based on neural logic programming [157] and neural theorem provers [107], which are able to learn logic rules and their weights simultaneously in a differentiable way. Though empirically effective for prediction, the search space of these methods is exponentially large, making it hard to identify high-quality logic rules. Besides, some recent efforts [153] formulate the problem as a sequential decision making process, and use reinforcement learning to search for logic rules, 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 logic rules which are used by the reasoning predictor for reasoning, 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., $\boldsymbol{q} = (h,\mathtt{r},?)$ and $\boldsymbol{a} = t$, we model the probability of the answer conditioned on query and existing knowledge graph $\mathcal{G}$, i.e., $p(\boldsymbol{a}|\mathcal{G},\boldsymbol{q})$, where a set of logic rules $\boldsymbol{z}$ [1] is treated as a latent variable. The rule generator defines a prior distribution over logic rules for each query, i.e., $p(\boldsymbol{z}|\boldsymbol{q})$, which is parameterized by a recurrent neural network. The reasoning predictor computes the likelihood of the answer conditioned on the logic rules and the existing knowledge graph $\mathcal{G}$, i.e., $p(\boldsymbol{a}|\mathcal{G},\boldsymbol{q},\boldsymbol{z})$. At each training iteration, we first sample a few logic rules from the rule generator, and further update the reasoning predictor to try out these rules for prediction. Then an EM algorithm [89] is used to optimize the rule generator. In the E-step, a set of high-quality logic rules are selected from all the generated rules according to their posterior probabilities. In the M-step, the rule generator is updated to imitate the high-quality rules selected in the E-step. Extensive experimental results show that RNNLogic outperforms state-of-the-art methods for knowledge graph reasoning [2]. Besides, RNNLogic is able to generate high-quality logic rules.

[1] More precisely, $\boldsymbol{z}$ is a multiset. In this paper, we use "set" to refer to "multiset" for conciseness.

[2] The codes of RNNLogic are available: `https://github.com/DeepGraphLearning/RNNLogic`

6.2. Related Work

Our work is related to existing efforts on learning logic rules for knowledge graph reasoning. Most traditional methods enumerate relational paths between query entities and answer entities as candidate logic rules, and further learn a scalar weight for each rule to assess the quality. Representative methods include Markov logic networks [57, 62, 106], relational dependency networks [88, 90], rule mining algorithms [32, 82], path ranking [71, 72] and probabilistic personalized page rank (ProPPR) algorithms [143–145]. Some recent methods extend the idea by simultaneously learning logic rules and the weights in a differentiable way, and most of them are based on neural logic programming [18, 107, 108, 157, 158] or neural theorem provers [84, 107]. These methods and our approach are similar in spirit, as they are all able to learn the weights of logic rules efficiently. However, these existing methods try to simultaneously learn logic rules and their weights, which is nontrivial in terms of optimization. The main innovation of our approach is to separate rule generation and rule weight learning by introducing a rule generator and a reasoning predictor respectively, which can mutually enhance each other. The rule generator generates a few high-quality logic rules, and the reasoning predictor only focuses on learning the weights of such high-quality rules, which significantly reduces the search space and leads to better reasoning results. Meanwhile, the reasoning predictor can in turn help identify some useful logic rules to improve the rule generator.

The other kind of rule learning method is based on reinforcement learning. The general idea is to train pathfinding agents, which search for reasoning paths in knowledge graphs to answer questions, and then extract logic rules from reasoning paths [13, 23, 76, 117, 153]. However, training effective pathfinding agents is highly challenging, as the reward signal (i.e., whether a path ends at the correct answer) can be extremely sparse. Although some studies [76] try to get better reward by using embedding-based methods for reward shaping, the performance is still worse than most embedding-based methods. In our approach, the rule generator has a similar role to those pathfinding agents. The major difference is that we simultaneously train the rule generator and a reasoning predictor with logic rules, which mutually enhance each other. The reasoning predictor provides effective reward for training the rule generator, and the rule generator offers high-quality rules to improve the reasoning predictor.

Our work is also related to knowledge graph embedding, which solves knowledge graph reasoning by learning entity and relation embeddings in latent spaces [4, 8, 11, 26, 92, 121, 135, 146, 156]. With proper architectures, these methods are able to learn some simple logic rules. For example, TransE [8] can learn some composition rules. RotatE [121] can mine some composition rules, symmetric rules and inverse rules. However, these methods can only

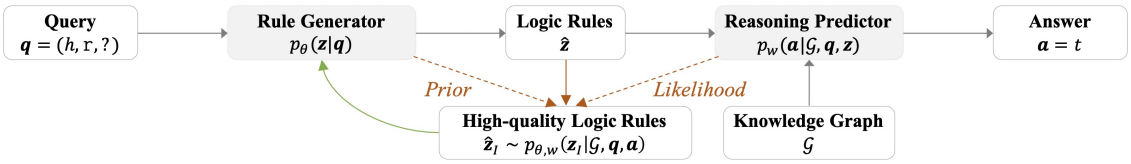


Figure 10. Overview of RNNLogic.

find some simple rules in an implicit way. In contrast, our approach explicitly trains a rule generator, which is able to generate more complicated logic rules.

There are some works studying boosting rule-based models [28, 43], where they dynamically add new rules according to the rule weights learned so far. These methods have been proven effective in binary classification and regression. Compared with them, our approach shares similar ideas, as we dynamically update the rule generator with the feedback from the reasoning predictor, but we focus on a different task, i.e., reasoning on knowledge graphs.

6.3. Preliminary

6.3.1. Problem Definition

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

6.3.2. Logic Rules

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 $\forall\{X, Y\}\ \mathtt{r}(X,Y) \leftarrow \mathtt{r}^{-1}(X,Y)$.

6.4. Model

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-based algorithm. In each iteration, we start with updating the

reasoning predictor to try out some logic rules generated by the rule generator. Then in the E-step, we identify a set of high-quality rules from all generated rules via posterior inference, with the prior from the rule generator and likelihood from the reasoning predictor. Finally in the M-step, the rule generator is updated with the identified high-quality rules.

RNNLogic has a rule generator p_θ and a reasoning predictor p_w. Given a query, the rule generator generates logic rules for the reasoning predictor. The reasoning predictor takes the generated rules as input, and reasons on a knowledge graph to predict the answer. RNNLogic is optimized with an EM-based algorithm. In each iteration, the rule generator produces some logic rules, and we update the reasoning predictor to explore these rules for reasoning. Then in the E-step, a set of high-quality rules are identified from all generated rules via posterior inference. Finally in the M-step, the rule generator is updated to be consistent with the high-quality rules identified in E-step.

6.4.1. Probabilistic Formalization

We start by formalizing knowledge graph reasoning in a probabilistic way, where a *set of logic rules* $\boldsymbol{z}$ is treated as a latent variable. The target distribution $p(\boldsymbol{a}|G, \boldsymbol{q})$ 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 $\boldsymbol{z}$ conditioned on a query $\boldsymbol{q}$, while the reasoning predictor p_w gives the likelihood of the answer $\boldsymbol{a}$ conditioned on latent rules $\boldsymbol{z}$, the query $\boldsymbol{q}$, and the knowledge graph G. Thus $p(\boldsymbol{a}|G, \boldsymbol{q})$ can be computed as below:

$$p_{w,\theta}(\boldsymbol{a}|G, \boldsymbol{q}) = \sum_{\boldsymbol{z}} p_w(\boldsymbol{a}|G, \boldsymbol{q}, \boldsymbol{z}) p_\theta(\boldsymbol{z}|\boldsymbol{q}) = \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{q})}[p_w(\boldsymbol{a}|G, \boldsymbol{q}, \boldsymbol{z})]. \tag{6.4.1}$$

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:

$$\max_{\theta,w} O(\theta, w) = \mathbb{E}_{(G,\boldsymbol{q},\boldsymbol{a})\sim p_{\text{data}}}[\log p_{w,\theta}(\boldsymbol{a}|G, \boldsymbol{q})]. \tag{6.4.2}$$

6.4.2. Parameterization

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

Formally, given a query $\boldsymbol{q} = (h,\mathtt{r},?)$, we generate compositional logic rules by only considering the query relation $\mathtt{r}$ without the query entity h, which allows the generated rules to generalize across entities. For each compositional rule in the abbreviation form $\mathtt{r} \leftarrow \mathtt{r}_1 \wedge \cdots \wedge \mathtt{r}_l$, it can be viewed as a sequence of 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 rule head, $\{\mathtt{r}_i\}_{i=1}^l$ 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 [49], and thus we introduce RNN_θ to parameterize the rule generator. Given a query relation $\mathtt{r}$, RNN_θ sequentially generates each relation in the body of a rule, until it reaches the ending relation $\mathtt{r}_{\text{END}}$. In this process, the probabilities of generated rules are simultaneously computed. With such rule probabilities, we define the distribution over a set of rules $\boldsymbol{z}$ as a multinomial distribution:

$$p_\theta(\boldsymbol{z}|\boldsymbol{q}) = \text{Mu}(\boldsymbol{z}|N, \text{RNN}_\theta(\cdot|\mathtt{r})), \tag{6.4.3}$$

where Mu stands for multinomial distributions, N is a hyperparameter for the size of the set $\boldsymbol{z}$, and $\text{RNN}_\theta(\cdot|\mathtt{r})$ defines a distribution over compositional rules with rule head being $\mathtt{r}$. The generative process of a rule set $\boldsymbol{z}$ is quite intuitive, where we simply generate N rules with RNN_θ to form $\boldsymbol{z}$.

Reasoning Predictor with Logic Rules. The reasoning predictor defines $p_w(\boldsymbol{a}|G, \boldsymbol{q}, \boldsymbol{z})$. For a query $\boldsymbol{q}$, the predictor uses a set of rules $\boldsymbol{z}$ to reason on a knowledge graph G and predict the answer $\boldsymbol{a}$.

Following stochastic logic programming [21], a principled reasoning framework, we use a log-linear model for reasoning. For each query $\boldsymbol{q} = (h,\mathtt{r},?)$, a compositional rule is able to find different grounding paths on graph G, leading to different candidate answers. For example, given query $(\textit{Alice}, \mathtt{hobby}, ?)$, a rule $\mathtt{hobby} \leftarrow \mathtt{friend} \wedge \mathtt{hobby}$ can have two groundings, $\mathtt{hobby}(\textit{Alice},\textit{Sing}) \leftarrow \mathtt{friend}(\textit{Alice},\textit{Bob}) \wedge \mathtt{hobby}(\textit{Bob},\textit{Sing})$ and $\mathtt{hobby}(\textit{Alice},\textit{Ski}) \leftarrow \mathtt{friend}(\textit{Alice},\textit{Charlie}) \wedge \mathtt{hobby}(\textit{Charlie},\textit{Ski})$, yielding two candidate answers *Sing* and *Ski*.

Let A be the set of candidate answers which can be discovered by any logic rule in the set $\boldsymbol{z}$. For each candidate answer $e \in A$, we compute the following scalar $\mathtt{score}_w(e)$ for that candidate:

$$\mathtt{score}_w(e) = \sum_{rule \in \boldsymbol{z}} \mathtt{score}_w(e|rule) = \sum_{rule \in \boldsymbol{z}} \sum_{path \in P(h,rule,e)} \psi_w(rule) \cdot \phi_w(path), \tag{6.4.4}$$

where $P(h,rule,e)$ is the set of grounding paths which start at h and end at e following a *rule* (e.g., $\textit{Alice} \xrightarrow{\mathtt{friend}} \textit{Bob} \xrightarrow{\mathtt{hobby}} \textit{Sing}$). $\psi_w(rule)$ and $\phi_w(path)$ are scalar weights of each *rule* and *path*. Intuitively, the score of each candidate answer e is the sum of scores contributed by each rule, i.e., $\mathtt{score}_w(e|rule)$. To get $\mathtt{score}_w(e|rule)$, we sum over every grounding *path* found in the graph G.

For the scalar weight $\psi_w(rule)$ of each *rule*, it is a learnable parameter to optimize. For the score $\phi_w(path)$ of each specific *path*, we explore two methods for parameterization. One method always sets $\phi_w(path) = 1$. However, this method cannot distinguish between different relational paths. To address the problem, for the other method, we follow an embedding algorithm RotatE [121] to introduce an embedding for each entity and model each relation

as a rotation operator on entity embeddings. Then for each grounding *path* of *rule* starting from h to e, if we rotate the embedding of h according to the rotation operators defined by the body relations of *rule*, we should expect to obtain an embedding close to the embedding of e. Thus we compute the similarity between the derived embedding and the embedding of e as $\phi_w(path)$, which can be viewed as a measure of the soundness and consistency of each *path*. For example, given a path *Alice* $\xrightarrow{\texttt{friend}}$ *Bob* $\xrightarrow{\texttt{hobby}}$ *Sing*, we rotate *Alice*'s embedding with the operators defined by `friend` and `hobby`. Afterwards we compute the similarity of the derived embedding and embedding of *Sing* as $\phi_w(path)$. Such a method allows us to compute a specific $\phi_w(path)$ for each *path*, which further leads to more precise scores for candidate answers.

Once we have the score for every candidate answer, we can further define the probability that the answer $\boldsymbol{a}$ of the query $\boldsymbol{q}$ is entity e by using a softmax function as follows:

$$p_w(\boldsymbol{a} = e|G,\boldsymbol{q},\boldsymbol{z}) = \frac{\exp(\texttt{score}_w(e))}{\sum_{e' \in A} \exp(\texttt{score}_w(e'))}. \tag{6.4.5}$$

6.4.3. Optimization

Next, we introduce how we optimize the reasoning predictor and rule generator to maximize the objective in Equation (6.4.2). At each training iteration, we first update the reasoning predictor p_w according to some rules generated by the generator, and then update the rule generator p_θ with an EM algorithm. In the E-step, a set of high-quality rules are identified from all generated rules via posterior inference, with the rule generator as the prior and the reasoning predictor as the likelihood. In the M-step, the rule generator is then updated to be consistent with the high-quality rules selected in the E-step.

Formally, at each training iteration, we start with maximizing the objective $O(\theta, w)$ in Equation (6.4.2) with respect to the reasoning predictor p_w. To do that, we notice that there is an expectation operation with respect to $p_\theta(\boldsymbol{z}|\boldsymbol{q})$ for each training instance $(G, \boldsymbol{q}, \boldsymbol{a})$. By drawing a sample $\hat{\boldsymbol{z}} \sim p_\theta(\boldsymbol{z}|\boldsymbol{q})$ for query $\boldsymbol{q}$, we can approximate the objective function of w at each training instance $(G, \boldsymbol{q}, \boldsymbol{a})$ as below:

$$O_{(G,\boldsymbol{q},\boldsymbol{a})}(w) = \log \mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{q})}[p_w(\boldsymbol{a}|G,\boldsymbol{q},\boldsymbol{z})] \approx \log p_w(\boldsymbol{a}|G,\boldsymbol{q},\hat{\boldsymbol{z}}) \tag{6.4.6}$$

Basically, we sample a set of rules $\hat{\boldsymbol{z}}$ from the generator and feed $\hat{\boldsymbol{z}}$ into the reasoning predictor. Then the parameter w of the reasoning predictor is updated to maximize the log-likelihood of the answer $\boldsymbol{a}$.

With the updated reasoning predictor, we then update the rule generator p_θ to maximize the objective $O(\theta, w)$ by using an EM framework.

E-step. Recall that when optimizing the reasoning predictor, we draw a set of rules $\hat{\boldsymbol{z}}$ for each data instance $(G, \boldsymbol{q}, \boldsymbol{a})$, and let the reasoning predictor use $\hat{\boldsymbol{z}}$ to predict $\boldsymbol{a}$. For each

data instance, the E-step aims to identify a set of K high-quality rules $\boldsymbol{z}_I$ from all generated rules $\hat{\boldsymbol{z}}$, i.e., $\boldsymbol{z}_I \subset \hat{\boldsymbol{z}}, |\boldsymbol{z}_I| = K$.

Formally, this is achieved by considering the posterior probability of each subset of logic rules $\boldsymbol{z}_I$, i.e., $p_{\theta,w}(\boldsymbol{z}_I|G, \boldsymbol{q}, \boldsymbol{a}) \propto p_w(\boldsymbol{a}|G, \boldsymbol{q}, \boldsymbol{z}_I)p_\theta(\boldsymbol{z}_I|\boldsymbol{q})$, with prior of $\boldsymbol{z}_I$ from the rule generator p_θ and likelihood from the reasoning predictor p_w. The posterior combines knowledge from both the rule generator and reasoning predictor, so the likely set of high-quality rules can be obtained by sampling from the posterior. However, sampling from the posterior is nontrivial due to its intractable partition function, so we approximate the posterior using a more tractable form with the proposition below:

Proposition 2. *For an instance $(G, \boldsymbol{q}, \boldsymbol{a})$ with $\boldsymbol{q} = (h,r,?)$ and $\boldsymbol{a} = t$, and a set of rules $\hat{\boldsymbol{z}}$ generated by the rule generator p_θ, we compute the following score H for each rule $\in \hat{\boldsymbol{z}}$:*

$$H(rule) = \left\{ \texttt{score}_w(t|rule) - \frac{1}{|A|} \sum_{e \in A} \texttt{score}_w(e|rule) \right\} + \log \mathrm{RNN}_\theta(rule|r), \tag{6.4.7}$$

where A is the set of all candidate answers discovered by rules in $\hat{\boldsymbol{z}}$, $\texttt{score}_w(e|rule)$ is the score that each rule contributes to entity e as defined by Equation (6.4.4)*, $\mathrm{RNN}_\theta(rule|r)$ is the prior probability of rule computed by the generator. If $s = \max_{e \in A} |\texttt{score}_w(e)| < 1$, then for a subset $\boldsymbol{z}_I \subset \hat{\boldsymbol{z}}$ with $|\boldsymbol{z}_I| = K$, $\log p_{\theta,w}(\boldsymbol{z}_I|G, \boldsymbol{q}, \boldsymbol{a})$ can be approximated as:*

$$\left| \log p_{\theta,w}(\boldsymbol{z}_I|G, \boldsymbol{q}, \boldsymbol{a}) - \left(\sum_{rule \in \boldsymbol{z}_I} H(rule) + \gamma(\boldsymbol{z}_I) + \text{const} \right) \right| \leq s^2 + O(s^4) \tag{6.4.8}$$

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

We prove the proposition in Section D.1. In practice, we can apply weight decay to the weight of logic rules in Equation (6.4.4), and thereby reduce $s = \max_{e \in A} |\texttt{score}_w(e)|$ to get a more precise approximation. The above proposition allows us to use $(\sum_{rule \in \boldsymbol{z}_I} H(rule) + \gamma(\boldsymbol{z}_I) + \text{const})$ to approximate the log-posterior $\log p_{\theta,w}(\boldsymbol{z}_I|G, \boldsymbol{q}, \boldsymbol{a})$, yielding a distribution $q(\boldsymbol{z}_I) \propto \exp(\sum_{rule \in \boldsymbol{z}_I} H(rule) + \gamma(\boldsymbol{z}_I))$ as a good approximation of the posterior. It turns out that the derived $q(\boldsymbol{z}_I)$ is a multinomial distribution, and thus sampling from $q(\boldsymbol{z}_I)$ is more tractable. Specifically, a sample $\hat{\boldsymbol{z}}_I$ from $q(\boldsymbol{z}_I)$ can be formed with K logic rules which are independently sampled from $\hat{\boldsymbol{z}}$, where the probability of each *rule* is computed as $\exp(H(rule))/(\sum_{rule' \in \hat{\boldsymbol{z}}} \exp(H(rule')))$. We provide the proof in Section D.2.

Intuitively, we could view $H(rule)$ of each *rule* as an assessment of the rule quality, which considers two factors. The first factor is based on the reasoning predictor p_w, and it is computed as the score that a *rule* contributes to the correct answer t minus the mean score that this *rule* contributes to other candidate answers. If a rule gives higher score to the true answer and lower score to other candidate answers, then the rule is more likely to be

important. The second factor is based on the rule generator p_θ, where we compute the prior probability for each *rule* and use the probability for regularization.

Empirically, we find that picking K rules with highest $H(rule)$ to form $\hat{\boldsymbol{z}}_I$ works better than sampling from the posterior. In fact, $\hat{\boldsymbol{z}}_I$ formed by top-K rules is an MAP estimation of the posterior, and thus the variant of picking top-K rules yields a hard-assignment EM algorithm [64]. Despite the reduced theoretical guarantees, we use this variant in practice for its good performance.

M-step. Once we obtain a set of high-quality logic rules $\hat{\boldsymbol{z}}_I$ for each data instance $(G, \boldsymbol{q}, \boldsymbol{a})$ in the E-step, we further leverage those rules to update the parameters θ of the rule generator in the M-step.

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

$$O_{(G,\boldsymbol{q},\boldsymbol{a})}(\theta) = \log p_\theta(\hat{\boldsymbol{z}}_I|\boldsymbol{q}) = \sum_{rule \in \hat{\boldsymbol{z}}_I} \log \mathrm{RNN}_\theta(rule|r) + \text{const.} \tag{6.4.9}$$

With the above objective, the feedback from the reasoning predictor can be effectively distilled into the rule generator. In this way, the rule generator will learn to only generate high-quality rules for the reasoning predictor to explore, which reduces the search space and yields better empirical results.

Algorithm 2 Workflow of RNNLogic

---

while not converge **do**

For each instance, use rule generator p_θ to generate a set of rules $\hat{\boldsymbol{z}}$ ($|\hat{\boldsymbol{z}}| = N$).

For each instance, update the predictor p_w with generated rules $\hat{\boldsymbol{z}}$ and Equation (6.4.6).

⊡ *E-step:*

For each instance, identify top-K rules $\hat{\boldsymbol{z}}_I$ from $\hat{\boldsymbol{z}}$ based on $H(rule)$ in Equation (6.4.7).

⊡ *M-step:*

For each instance, update rule generator p_θ with the identified rules and Equation (6.4.9).

end while

During testing, use p_θ to generate rules and feed them into p_w for prediction.

Connection with REINFORCE. In terms of optimizing the rule generator, our EM algorithm has some connections with the REINFORCE algorithm [150]. Formally, for a training instance $(\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$, REINFORCE computes the derivative with respect to the parameters θ

of the rule generator as follows:

$$\mathbb{E}_{p_\theta(\boldsymbol{z}|\boldsymbol{q})}[R \cdot \nabla_\theta \log p_\theta(\boldsymbol{z}|\boldsymbol{q})] \simeq R \cdot \nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}|\boldsymbol{q}), \tag{6.4.10}$$

where $\hat{\boldsymbol{z}}$ is a sample from the rule generator, i.e., $\hat{\boldsymbol{z}} \sim p_\theta(\boldsymbol{z}|\boldsymbol{q})$. R is a reward from the reasoning predictor based on the prediction result on the instance. For example, we could treat the probability of the correct answer computed by the reasoning predictor as reward, i.e., $R = p_w(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q}, \hat{\boldsymbol{z}})$.

In contrast, our EM optimization algorithm optimizes the rule generator with the objective function defined in Equation (6.4.2), yielding the derivative $\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_I|\boldsymbol{q})$. Comparing the derivative to that in REINFORCE (Equation (6.4.10)), we see that EM only maximizes the log-probability for rules selected in the E-step, while REINFORCE maximizes the log-probability for all generated rules weighted by the scalar reward R. Hence the two approaches coincide if R is set to 1 for the selected rules from the approximate posterior and 0 otherwise. In general, finding an effective reward function to provide feedback is nontrivial, and we empirically compare these two optimization algorithms in experiments.

6.4.4. RNNLogic+

In RNNLogic, we aim at jointly training a rule generator and a simple reasoning predictor. Although this framework allows us to generate high-quality logic rules, the performance of knowledge graph reasoning is limited by the low capacity of the reasoning predictor. To further improve the results, a natural idea is to only focus on the reasoning predictor and develop a more powerful predictor by using the high-quality rules generated by RNNLogic as input. Next, we propose one such predictor.

Formally, let $\hat{\boldsymbol{z}}_I$ be the set of generated high-quality logic rules. Given a query $\boldsymbol{q} = (h,\mathtt{r},?)$, let $\mathcal{A}$ be the collection of candidate answers that can be discovered by any rule in $\hat{\boldsymbol{z}}_I$. For each candidate answer $e \in \mathcal{A}$, we again compute a scalar score $\mathtt{score}_w(e)$ for that candidate answer as below:

$$\mathtt{score}_w(e) = \mathrm{MLP}(\mathrm{AGG}(\{\mathbf{v}_{rule}, |\mathcal{P}(h,rule,e)|\}_{rule \in \hat{\boldsymbol{z}}_I}). \tag{6.4.11}$$

Here, $\mathbf{v}_{rule}$ is an embedding vector for each *rule*, and $|\mathcal{P}(h,rule,e)|$ is the number of grounding paths from h to e discovered by *rule*. AGG is an aggregator, which aims at aggregating all the rule embeddings $\mathbf{v}_{rule}$ by treating $|\mathcal{P}(h,rule,e)|$ as aggregation weights. We follow Zhu et al. [171] to use the PNA aggregator [19] for its good performance. Once we get the aggregated embedding, an MLP is further used to project the embedding to the scalar score for candidate e.

In practice, we can further enhance the above score with knowledge graph embedding. For each query $\boldsymbol{q} = (h,\mathtt{r},?)$ and candidate answer e, knowledge graph embedding methods

are able to infer a plausibility score $\text{KGE}(h, \mathtt{r}, e)$, measuring how likely $(h, \mathtt{r}, e)$ is a valid triplet. Based on that, we can naturally obtain a more powerful score function by combining the score from logic rules and the score from knowledge graph embedding. For example, we can linearly combine them as follows:

$$\mathtt{score}_w(e) = \text{MLP}(\text{AGG}(\{\mathbf{v}_{rule}, |\mathcal{P}(h,rule,e)|\}_{rule \in \hat{\boldsymbol{z}}_I}) + \eta \ \text{KGE}(h, \mathtt{r}, e), \tag{6.4.12}$$

where η controls the weight of the knowledge graph embedding score.

Once we have the score $\mathtt{score}_w(e)$ for each candidate e, we can again apply a softmax function to the score as in Equation (6.4.5) to compute the probability that e is the answer, i.e., $p_w(\boldsymbol{a} = e|\mathcal{G},\boldsymbol{q},\boldsymbol{z})$. This predictor can then be easily optimized through maximizing likelihood on each instance $(\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$.

6.5. Experiment

6.5.1. Settings

Datasets. We choose four datasets for evaluation, including FB15k-237 [132], WN18RR [26], Kinship and UMLS [63]. For Kinship and UMLS, there are no standard data splits, so we randomly sample 30% of all the triplets for training, 20% for validation, and the rest 50% for testing. The detailed statistics are summarized in Section D.5.

Compared Algorithms. We compare the following algorithms in experiment:

⊡ *Rule learning methods.* For traditional methods in statistical relational learning, we choose Markov logic networks [106], boosted relational dependency networks [88] and path ranking [71]. We also consider some methods based on neural logic programming, including NeuralLP [157], DRUM [108] and NLIL [158]. In addition, we compare against CTP [84], a differentiable method based on neural theorem provers. Besides, two reinforcement learning methods are compared, which are MINERVA [23] and M-Walk [117].

⊡ *Other methods.* We also consider some embedding methods, including TransE [8], DistMult [156], ComplEx [135], ComplEx-N3 [68], ConvE [26], TuckER [4], RotatE [121].

⊡ *RNNLogic.* For RNNLogic, we consider two model variants. The first variant assigns a constant score to different grounding paths in the reasoning predictor, i.e., $\phi_w(path) = 1$ in Equation (6.4.4), and we denote this variant as *w/o emb.*. The second variant leverages entity embeddings and relation embeddings to compute the path score $\phi_w(path)$, and we denote the variant as *with emb.*.

⊡ *RNNLogic+.* For RNNLogic+, we also consider two model variants. The first variant only uses the logic rules learned by RNNLogic to train the reasoning predictor as in Equation (6.4.11), and we denote the variant as *w/o emb.*. The second variant uses both of the

learned logic rules and knowledge graph embeddings to train a reasoning predictor as in Equation (6.4.12), and we denote the variant as *with emb.*.

Evaluation Metrics. During evaluation, for each test triplet $(h, \mathtt{r}, t)$, we build two queries $(h, \mathtt{r}, ?)$ and $(t, \mathtt{r}^{-1}, ?)$ with answers t and h. For each query, we compute a probability for each entity, and compute the rank of the correct answer. Given the ranks from all queries, we report the Mean Rank (MR), Mean Reciprocal Rank (MRR) and Hit@k (H@k) under the filtered setting [8], which is used by most existing studies. Note that there can be a case where an algorithm assigns the same probability to the correct answer and a few other entities. For such a case, many methods compute the rank of the correct answer as $(m + 1)$ where m is the number of entities receiving higher probabilities than the correct answer. This setup can be problematic according to Sun et al. [122]. For fair comparison, in that case we compute the expectation of each evaluation metric over all the random shuffles of entities which receive the same probability as the correct answer. For example, if there are n entities which have the same probability as the correct answer in the above case, then we treat the rank of the correct answer as $(m + (n + 1)/2)$ when computing Mean Rank.

Besides, we notice that in MINERVA [23] and MultiHopKG [76], they only consider queries in the form of $(h, \mathtt{r}, ?)$, which is different from our default setting. To make fair comparison with these methods, we also apply RNNLogic to this setting and report the performance.

Experimental Setup of RNNLogic. For each training triplet $(h, \mathtt{r}, t)$, we add an inverse triplet $(t, \mathtt{r}^{-1}, h)$ into the training set, yielding an augmented set of training triplets $\mathcal{T}$. We use a closed-world assumption for model training, which assumes that any triplet outside $\mathcal{T}$ is incorrect. To build a training instance from p_{data}, we first randomly sample a triplet $(h, \mathtt{r}, t)$ from $\mathcal{T}$, and then form an instance as $(\mathcal{G} = \mathcal{T} \setminus \{(h, \mathtt{r}, t)\}, \boldsymbol{q} = (h, \mathtt{r}, ?), \boldsymbol{a} = t)$. Basically, we use the sampled triplet $(h, \mathtt{r}, t)$ to construct the query and answer, and use the rest of triplets in $\mathcal{T}$ to form the background knowledge graph $\mathcal{G}$. During testing, the background knowledge graph $\mathcal{G}$ is formed with all the triplets in $\mathcal{T}$.

For the rule generator of RNNLogic, the maximum length of generated rules is set to 4 for FB15k-237, 5 for WN18RR, and 3 for the rest, which are selected on validation data. See Section D.5 for the details.

Once RNNLogic is trained, we leverage the rule generator to generate a few high-quality logic rules, which are further utilized to train RNNLogic+. Specifically, the maximum length of generated rules is set to 3. We generate 100 rules for each relation in FB15k-237 and 200 rules for each relation in WN18RR. The dimension of logic rule embedding in RNNLogic+ is set to 32 on all datasets. The hyperparameter η in Equation (6.4.12) is set to 2 on the FB15k-237 dataset and 0.5 on the WN18RR dataset.

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
	ComplEx-N3*	-	**0.37**	-	-	**56**	-	0.48	-	-	57
	ConvE*	244	0.325	23.7	35.6	50.1	4187	0.43	40	44	52
	TuckER*	-	0.358	**26.6**	**39.4**	54.4	-	0.470	44.3	48.2	52.6
	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	-	-	-	-	-
	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
RNNLogic+	w/o emb.	480	0.299	21.5	32.8	46.4	7204	0.489	45.3	50.6	56.3
	with emb.	178	0.349	25.8	38.5	53.3	4624	**0.513**	**47.1**	**53.2**	**59.7**

Table 25. Results of RNNLogic on FB15k-237 and WN18RR.

Category	Algorithm	FB15k-237					WN18RR				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
Rule Learning	MINERVA*	-	0.293	21.7	32.9	45.6	-	0.448	41.3	45.6	51.3
	MultiHopKG*	-	0.407	32.7	-	56.4	-	0.472	43.7	-	54.2
RNNLogic	w/o emb.	459.0	0.377	28.9	41.2	54.9	7662.8	0.478	43.8	50.3	55.3
	with emb.	**146.1**	**0.443**	**34.4**	**48.9**	**64.0**	**3767.0**	**0.506**	**46.3**	**52.3**	**59.2**

Table 26. Results of RNNLogic on FB15k-237 and WN18RR with only $(h, \mathtt{r}, ?)$-queries.

In RNNLogic+ *with emb.*, we use RotatE [121] as the knowledge graph embedding model, which is the same as RNNLogic *with emb.*. For both models, the entity and relation embeddings are pre-trained. The embedding dimension is set to 500 on FB15k-237 and 200 on WN18RR.

6.5.2. Results

Comparison against Existing Methods. We present the results on the FB15k-237 and WN18RR datasets in Table 25 and Table 26. The results on the Kinship and UMLS datasets are shown in Table 27. In the tables, H@k is in %. [*] means the numbers are taken from the original papers. [†] means we rerun the methods with the same evaluation process.

We first compare RNNLogic with rule learning methods. RNNLogic achieves much better results than statistical relational learning methods (MLN, Boosted RDN, PathRank) and neural differentiable methods (NeuralLP, DRUM, NLIL, CTP). This is because the rule generator and reasoning predictor of RNNLogic can collaborate with each other to reduce search space and learn better rules. RNNLogic also outperforms reinforcement learning

Category	Algorithm	Kinship					UMLS				
		MR	MRR	H@1	H@3	H@10	MR	MRR	H@1	H@3	H@10
No Rule Learning	DistMult	8.5	0.354	18.9	40.0	75.5	14.6	0.391	25.6	44.5	66.9
	ComplEx	7.8	0.418	24.2	49.9	81.2	13.6	0.411	27.3	46.8	70.0
	ComplEx-N3	-	0.605	43.7	71.0	92.1	-	0.791	68.9	87.3	95.7
	TuckER	6.2	0.603	46.2	69.8	86.3	5.7	0.732	62.5	81.2	90.9
	RotatE	3.7	0.651	50.4	75.5	93.2	4.0	0.744	63.6	82.2	93.9
Rule Learning	MLN	10.0	0.351	18.9	40.8	70.7	7.6	0.688	58.7	75.5	86.9
	Boosted RDN	25.2	0.469	39.5	52.0	56.7	54.8	0.227	14.7	25.6	37.6
	PathRank	-	0.369	27.2	41.6	67.3	-	0.197	14.8	21.4	25.2
	NeuralLP	16.9	0.302	16.7	33.9	59.6	10.3	0.483	33.2	56.3	77.5
	DRUM	11.6	0.334	18.3	37.8	67.5	8.4	0.548	35.8	69.9	85.4
	MINERVA	-	0.401	23.5	46.7	76.6	-	0.564	42.6	65.8	81.4
	CTP	-	0.335	17.7	37.6	70.3	-	0.404	28.8	43.0	67.4
RNNLogic	w/o emb.	3.9	0.639	49.5	73.1	92.4	5.3	0.745	63.0	83.3	92.4
	with emb.	**3.1**	**0.722**	**59.8**	**81.4**	**94.9**	**3.1**	**0.842**	**77.2**	**89.1**	**96.5**

Table 27. Results of RNNLogic on the Kinship and UMLS datasets.

methods (MINERVA, MultiHopKG, M-Walk). The reason is that RNNLogic is optimized with an EM-based framework, in which the reasoning predictor provides more useful feedback to the rule generator, and thus addresses 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 quite limited.

Results of RNNLogic+. In RNNLogic, we jointly train a rule generator and a simple rule generator. In contrast to that, RNNLogic+ focuses on training a complicated predictor with the high-quality rules learned by RNNLogic as input. In this way, RNNLogic+ entails high model capacity. Next, we look into RNNLogic+ and we present its results on the FB15k-237 and WN18RR datasets in Table 25.

We can see that without the help of knowledge graph embedding (*w/o emb.*), RNNLogic+ already gets competitive results on both datasets, especially WN18RR, where RNNLogic+ outperforms all the other methods. Surprisingly, these results are achieved by using only a few logic rules for prediction (100 for each relation in FB15k-237 and 200 for each relation in WN18RR). This observation shows that the logic rules learned by RNNLogic are indeed very useful. For now, the reasoning predictor used in RNNLogic+ is still straightforward, and we are likely to achieve better results by designing a more powerful predictor to make better use of the high-quality rules. We leave it as a future work.

By further using knowledge graph embedding (*with emb.*), the results of RNNLogic+ are significantly improved on both datasets. Moreover, RNNLogic+ with embedding also outperforms most knowledge graph embedding methods. On WN18RR, the method even achieves the best result. This demonstrates that the information captured by logic rules and knowledge graph embedding is complementary, allowing rule-based methods and embedding-based methods to mutually enhance each other. Thus, an interesting feature direction could be designing approaches to better combine both kinds of methods.

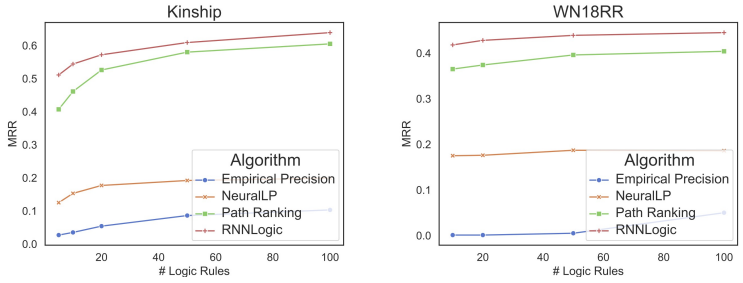


Figure 11. Performance w.r.t. the number of logic rules in RNNLogic.

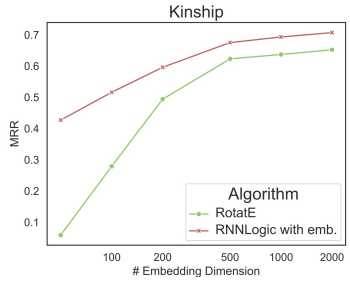


Figure 12. Results w.r.t. embedding dim. in RNNLogic.

Quality of Learned Logic Rules. Next, we study the quality of rules learned by different methods for reasoning. For each trained model, we let it generate I rules with highest qualities per query relation, and use them to train a predictor *w/o emb.* as in Equation (6.4.5) for reasoning. For RNNLogic, the quality of each rule is measured by its prior probability from the rule generator, and we use beam search to infer top-I rules. The results at different I are in Figure 11, where RNNLogic achieves much better results. Even with only 10 rules per relation, RNNLogic still achieves competitive results.

Performance w.r.t. the Number of Training Triplets. To better evaluate different methods under cases where training triplets are very limited, in this section we reduce the amount of training data on Kinship and UMLS to see how the performance varies. The results are presented in Figure 13. We see that RNNLogic *w/o emb.* achieves the best results. Besides, the improvement over RotatE is more significant as we reduce training triplets, showing that RNNLogic is more robust to data sparsity.

Performance w.r.t. Embedding Dimension. RNNLogic *with emb.* uses entity and relation embeddings to improve the reasoning predictor. Next, we study its performance with different embedding dimensions. The results are presented in Figure 12, where we compare against RotatE [121]. We see that RNNLogic significantly outperforms RotatE at

every embedding dimension. The improvement is mainly from the use of logic rules, showing that our learned rules are indeed helpful.

Comparison of Optimization Algorithms. RNNLogic uses an EM algorithm to optimize the rule generator. In practice, the generator can also be optimized with REINFORCE [150] (see Equation (6.4.10) for details). We empirically compare the two algorithms in the *w/o emb.* case. The results on Kinship and UMLS are presented in Table 28. We see EM consistently outperforms REINFORCE.

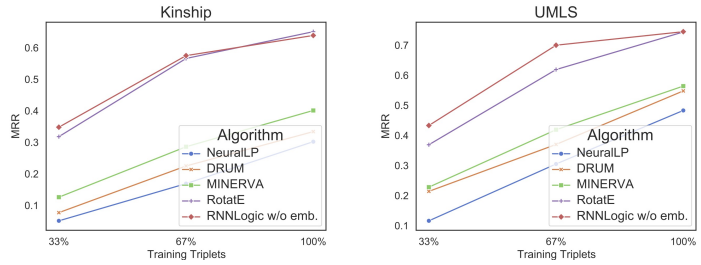


Figure 13. Performance w.r.t. the number of training triplets in RNNLogic.

	Kinship MRR	**UMLS MRR**
REINFORCE	0.312	0.504
EM	0.639	0.745

Table 28. Comparison of REINFORCE and EM in RNNLogic.

Case Study of Generated Logic Rules. Finally, we show some logic rules generated by RNNLogic on the FB15k-237 dataset in Table 29. We can see that these logic rules are meaningful and diverse. The first rule is a subrelation rule. The third and fifth rules are two-hop compositional rules. The rest of logic rules have even more complicated forms. This case study shows that RNNLogic can indeed learn useful and diverse rules for reasoning. More generated logic rules are available at Table 30. In this table, $h \xrightarrow{\mathtt{r}} t$ means triplet $(h,\mathtt{r},t)$ and $h \xleftarrow{\mathtt{r}} t$ means triplet $(h,\mathtt{r}^{-1},t)$, or equivalently $(t,\mathtt{r},h)$.

$\mathtt{Appears\_in\_TV\_Show}(X,Y) \leftarrow \mathtt{Actor\_of}(X,Y)$
$\mathtt{Appears\_in\_TV\_Show}(X,Y) \leftarrow \mathtt{Creator\_of}(X,U) \wedge \mathtt{Has\_Producer}(U,V) \wedge \mathtt{Appears\_in\_TV\_Show}(V,Y)$
$\mathtt{ORG.\_in\_State}(X,Y) \leftarrow \mathtt{ORG.\_in\_City}(X,U) \wedge \mathtt{City\_Locates\_in\_State}(U,Y)$
$\mathtt{ORG.\_in\_State}(X,Y) \leftarrow \mathtt{ORG.\_in\_City}(X,U) \wedge \mathtt{Address\_of\_PERS.}(U,V) \wedge \mathtt{Born\_in}(V,W) \wedge \mathtt{Town\_in\_State}(W,Y)$
$\mathtt{Person\_Nationality}(X,Y) \leftarrow \mathtt{Born\_in}(X,U) \wedge \mathtt{Place\_in\_Country}(U,Y)$
$\mathtt{Person\_Nationality}(X,Y) \leftarrow \mathtt{Student\_of\_Educational\_Institution}(X,U) \wedge \mathtt{ORG.\_Endowment\_Currency}(U,V) \wedge \mathtt{Currency\_Used\_in\_Region}(V,W) \wedge \mathtt{Region\_in\_Country}(W,Y)$

Table 29. Case study of the rules generated by the rule generator.

6.6. Conclusion

This paper studies learning logic rules for knowledge graph reasoning, and an approach called RNNLogic is proposed. RNNLogic treats a set of logic rules as a latent variable, and a rule generator as well as a reasoning predictor with logic rules are jointly learned. We develop

an EM-based algorithm for optimization. Extensive expemriments prove the effectiveness of RNNLogic. In the future, we plan to study generating more complicated logic rules rather than only compositional rules. Besides, we plan to extend RNNLogic to other reasoning problems, such as question answering.

Relation	←	Rule (*Explanation*)
$X \xrightarrow{\texttt{Appears\_in\_TV\_Show}} Y$	←	$X \xleftarrow{\texttt{Has\_Actor}} Y$
		(*Definition. An actor of a show appears in the show, obviously.*)
	←	$X \xrightarrow{\texttt{Creator\_of}} U \xleftarrow{\texttt{Producer\_of}} V \xrightarrow{\texttt{Appears\_in\_TV\_Show}} Y$
		(*The creator X and the producer V of another show U are likely to appear in the same show Y.*)
	←	$X \xleftarrow{\texttt{Actor\_of}} U \xleftarrow{\texttt{Award\_Nominated}} V \xleftarrow{\texttt{Winner\_of}} Y$
	←	$X \xrightarrow{\texttt{Writer\_of}} U \xleftarrow{\texttt{Creater\_of}} V \xrightarrow{\texttt{Actor\_of}} Y$
	←	$X \xrightarrow{\texttt{Student\_of}} U \xleftarrow{\texttt{Student\_of}} V \xrightarrow{\texttt{Appears\_in\_TV\_Show}} Y$
		(*Two students X and V in the same school U are likely to appear in the same show Y.*)
$X \xrightarrow{\texttt{ORG.\_in\_State}} Y$	←	$X \xrightarrow{\texttt{ORG.\_in\_City}} U \xrightarrow{\texttt{City\_in\_State}} Y$
		(*Use the city to indicate the state directly.*)
	←	$X \xrightarrow{\texttt{ORG.\_in\_City}} U \xleftarrow{\texttt{Lives\_in}} V \xrightarrow{\texttt{Born\_in}} W \xrightarrow{\texttt{Town\_in\_State}} Y$
		(*Use the person living in the city to induct the state.*)
	←	$X \xleftarrow{\texttt{Sub-ORG.\_of}} U \xrightarrow{\texttt{ORG.\_in\_State}} Y$
	←	$X \xrightarrow{\texttt{Sub-ORG.\_of}} U \xleftarrow{\texttt{Sub-ORG.\_of}} V \xrightarrow{\texttt{ORG.\_in\_State}} Y$
	←	$X \xrightarrow{\texttt{ORG.\_in\_City}} U \xleftarrow{\texttt{ORG.\_in\_City}} V \xrightarrow{\texttt{ORG.\_in\_State}} Y$
		(*Two organizations in the same city are in the same state.*)
$X \xrightarrow{\texttt{Person\_Nationality}} Y$	←	$X \xrightarrow{\texttt{Born\_in}} U \xrightarrow{\texttt{Place\_in\_Country}} Y$
		(*Definition.*)
	←	$X \xrightarrow{\texttt{Spouse}} U \xrightarrow{\texttt{Person\_Nationality}} Y$
		(*By a fact that people are likely to marry a person of same nationality.*)
	←	$X \xrightarrow{\texttt{Student\_of}} U \xrightarrow{\texttt{ORG.\_Endowment\_Currency}} V \xleftarrow{\texttt{Region\_Currency}} W \xrightarrow{\texttt{Region\_in\_Country}} Y$
		(*Use the currency to induct the nationality.*)
	←	$X \xrightarrow{\texttt{Born\_in}} U \xleftarrow{\texttt{Born\_in}} V \xrightarrow{\texttt{Person\_Nationality}} Y$
	←	$X \xrightarrow{\texttt{Politician\_of}} U \xleftarrow{\texttt{Politician\_of}} V \xrightarrow{\texttt{Person\_Nationality}} Y$
$X \xrightarrow{\texttt{Manifestation\_of}} Y$	←	$X \xleftarrow{\texttt{Treats}} U \xrightarrow{\texttt{Prevents}} V \xleftarrow{\texttt{Precedes}} Y$
	←	$X \xleftarrow{\texttt{Complicates}} U \xleftarrow{\texttt{Precedes}} Y$
	←	$X \xrightarrow{\texttt{Location\_of}} U \xrightarrow{\texttt{Is\_a}} V \xleftarrow{\texttt{Precedes}} Y$
	←	$X \xleftarrow{\texttt{Complicates}} U \xrightarrow{\texttt{Precedes}} V \xleftarrow{\texttt{Occurs\_in}} Y$
	←	$X \xrightarrow{\texttt{Location\_of}} U \xleftarrow{\texttt{Occurs\_in}} V \xleftarrow{\texttt{Occurs\_in}} Y$
	←	$X \xrightarrow{\texttt{Precedes}} U \xleftarrow{\texttt{Occurs\_in}} V \xleftarrow{\texttt{Degree\_of}} Y$
$X \xleftarrow{\texttt{Affects}} Y$	←	$X \xrightarrow{\texttt{Result\_of}} U \xrightarrow{\texttt{Occurs\_in}} V \xrightarrow{\texttt{Precedes}} Y$
	←	$X \xleftarrow{\texttt{Precedes}} U \xrightarrow{\texttt{Produces}} V \xleftarrow{\texttt{Occurs\_in}} Y$
	←	$X \xleftarrow{\texttt{Prevents}} U \xrightarrow{\texttt{Disrupts}} V \xrightarrow{\texttt{Co-occurs\_with}} Y$
	←	$X \xleftarrow{\texttt{Result\_of}} U \xrightarrow{\texttt{Complicates}} V \xrightarrow{\texttt{Precedes}} Y$
	←	$X \xleftarrow{\texttt{Assesses\_Effect\_of}} U \xrightarrow{\texttt{Method\_of}} V \xrightarrow{\texttt{Complicates}} Y$
	←	$X \xrightarrow{\texttt{Process\_of}} U \xrightarrow{\texttt{Interacts\_with}} V \xrightarrow{\texttt{Causes}} Y$
	←	$X \xleftarrow{\texttt{Assesses\_Effect\_of}} U \xleftarrow{\texttt{Result\_of}} V \xrightarrow{\texttt{Precedes}} Y$

Table 30. Logic rules learned by RNNLogic.

Chapter 7

DiffLogic: A Differentiable Approach of Rule Learning

This paper studies interpretable reasoning on knowledge graphs (a.k.a. link prediction) by learning symbolic logic rules, which is very critical in many applications such as recommender systems and biomedicine. Exiting methods either infer logic rules in an indirect manner through post-processing (e.g., NeurILP) or suffer from expensive computation with alternative optimization methods (e.g., RNNLogic). In this paper, we propose an end-to-end solution for directly learning logic rules from knowledge graphs. Specifically, we jointly train a policy network for explicit rule generation as well as a Transformer-based reasoning network for reasoning with generated logic rules, which enjoy high flexibility and interpretability. To train both networks in an end-to-end fashion, we design an optimization technique based on importance sampling, where a proposal distribution is introduced to probe different logic rules used for training, and the proposal distribution is dynamically updated to balance between exploitation and exploration. Extensive experiments on multiple benchmarks show the effectiveness of our proposed approach.

7.1. Introduction

Knowledge graphs are data structures used to store real-world knowledge, where each piece of knowledge is represented as a fact (h, r, t) or $r(h, t)$, meaning head entity h and tail entity t have relation `r`. These facts have been proven useful in many applications, including question answering [75], drug repurposing [155], and recommender systems [142]. However, as collecting all possible facts is impossible, knowledge graphs are inevitably incomplete. Therefore, how to infer missing facts in a knowledge graph by reasoning with existing facts (a.k.a., knowledge graph reasoning) has become an important task.

Recently, learning high-quality logic rules has received increasing attention for knowledge graph reasoning. For example, one might learn a rule $\forall X,Y,Z$ $\texttt{nationality}(X,Y) \leftarrow \texttt{born\_in}(X,Z) \wedge \texttt{capital\_of}(Z,Y)$, indicating if person X was born in city Z and city Z is in country Y, then the nationality of X would be Y. Through a few such rules, we can have through understanding of `nationality` and further apply these rules to infer new facts. With logic rules, we are able to develop models with high interpretability [101, 166] and generalizability [130, 171]. However, how to learn useful logic rules for reasoning on knowledge graphs remains a nontrivial task due to the huge search space of rules.

In literature, there are mainly two kinds of methods proposed for logic rule learning. One is the path-based method, which predicts the relation of two entities by enumerating and aggregating the relational paths between these entities. After training, high-quality logic rules can be extracted by post-processing these relational paths. For example, DeepPath [153] and MINERVA [23] use reinforcement learning to identify a few important relational paths for reasoning. GraIL [130] and NBFNet [171] employ nonlinear graph neural networks to aggregate all possible relational paths for reasoning. However, these methods learn logic rules indirectly by post-processing relational paths, and extracting universal logic rules from relational paths is nontrivial, meaning these methods lack sufficient interpretability. The other methods aim at generating logic rules directly. For example, classical rule mining methods [32, 82] search for logic rules which are consistent with the observed facts. A recent approach RNNLogic [105] leverages an LSTM model for generating logic rules. Despite the good interpretability, these methods lack an end-to-end training algorithm and different logic rules are combined in a simplistic manner.

Ideally, we would expect to develop a new approach which inherits the merits of both methods, one with high interpretability as well as good performance. In this paper, we propose such an approach. Our approach improves interpretability by using a policy network to directly generate logic rules. These logic rules are further fed into a Transformer-based graph reasoning network, which aggregates these rules for precise knowledge reasoning. By jointly training a policy network and a reasoning network, our approach enjoys both high interpretability and good performance. However, training them in an end-to-end fashion is nontrivial, as computing the gradient of both networks requires sampling from the intractable posterior distribution of logic rules. The prior comes from the policy network and the likelihood is from the reasoning network. We address the problem by using importance sampling, where a proposal distribution is used as a surrogate to the posterior distribution for probing different combinations of logic rules. During training, the proposal distribution is dynamically updated to balance between exploration of unseen logic rules and exploitation of

promising new ones. Extensive experiments on four standard datasets prove the effectiveness of our proposed approach.

7.2. Related Work

In knowledge graph reasoning, a variety of methods have been proposed for learning logic rules, either indirectly or directly. Indirect rule learning methods are mainly path-based. Their idea is to enumerate the relational paths between a pair of entities to predict their relation, and further extract high-quality logic rules by post-processing these relational paths. Classical path-based methods such as path ranking [71, 72] and ProPPR [143, 144, 146] combine relational paths with log-linear models for reasoning, yet the model capacity is limited, yielding unsatisfactory results. Some later efforts propose to use reinforcement learning to train pathfinding agents, which are able to identify a few important important paths for reasoning [13, 23, 76, 117, 153]. Nevertheless, as only a few relational paths are used for reasoning, these methods suffer from inferior results, and training the pathfinding agents is highly challenging due to sparse reward. More recent methods perform reasoning by aggregating all the relational paths between a pair of entities via graph neural networks [61, 137]. Representative methods include neural logic programming [18, 107, 108, 157, 158], GraIL [130], and NBFNet [171]. However, as these methods use highly nonlinear aggregators to combine numerous relational paths for reasoning, how to extract universal logic rules from these relational paths remains unclear, and thus these methods lack desired interpretability. Compared with all path-based methods, our approach employs a policy network to explicitly generate high-quality logic rules, which enjoys better interpretability. Besides, we develop a Transformer-based reasoning network to make use of logic rules, allowing our approach to achieve comparable results to state-of-the-art path-based methods.

The other kind of method directly generates logic rules for knowledge graph reasoning. Most methods are based on the idea of generate-and-test, i.e., generating candidate logic rules and testing whether they are consistent with observed facts. Traditional rule mining algorithms [32, 82] mine logic rules by measuring rule qualities with hand-crafted heuristics. Nevertheless, logic rules mined with such heuristics often suffer from poor results. One recent work RNNLogic [105] improves this idea by jointly training a rule generator and a reasoning predictor, where the rule generator produces logic rules for the reasoning predictor, which in turn provides feedback to update the rule generator. Our approach is in a similar vein to this work. However, this method uses EM algorithm for optimization, which cannot be executed in an end-to-end fashion and is only compatible with log-linear predictors, leading to worse results. In contrast, we develop an end-to-end training algorithm by using the idea of importance sampling, which generalizes to different kinds of reasoning models. Based on

that, we further equip our approach with a Transformer-based reasoning network, allowing the approach to have superior results.

Lastly, there also exist various embedding-based methods for knowledge graph reasoning, a.k.a., knowledge graph embedding models. Their idea is to learn useful entity and relation embeddings through defining score functions to discriminate between valid $(h, \mathtt{r}, t)$ facts and some noisy ones. With some well-designed score functions, knowledge graph embedding methods can inherently capture some simple rule patterns, and thus achieve impressive results on many datasets. Nevertheless, these methods are not able to explicitly learn logic rules and use them for knowledge graph reasoning, whereas our approach employs a policy network for rule generation, which has better interpretability.

7.3. Model

This paper focuses on learning logic rules for knowledge graph reasoning. Existing methods either suffer from insufficient interpretability due to indirect rule learning through post-processing, or lack an end-to-end algorithm to train high-capacity reasoning models. We address these limitations by proposing a principled approach. Our approach treats a set of logic rules as latent variables, and jointly train a policy network for generating high-quality logic rules as well as a Transformer-based network for reasoning with these logic rules. To effectively train the policy network and reasoning network in an end-to-end fashion, we propose to combine gradient descent and importance sampling. Specifically, a proposal distribution probes different sets of logic rules for training both networks, and the proposal distribution is dynamically updated to balance exploration with exploitation, so that both networks can be continuously improved.

Formally, the problem of knowledge graph reasoning involves a background knowledge graph $\mathcal{G}$, which contains all the observed facts. Then given a query $\mathbf{x} = (h, \mathtt{r}, ?)$ asking which entity has relation $\mathtt{r}$ with entity h, we aim to predict the answer entity $\mathbf{y}$. Therefore, the goal of knowledge graph reasoning can be formulated as modeling the conditional distribution $p(\mathbf{y}|\mathcal{G}, \mathbf{x})$.

Our approach solves the problem through treating logic rules as latent variables. Intuitively, given a query $\mathbf{x} = (h, \mathtt{r}, ?)$, we start with generating a collection of logic rules $\mathbf{z}$ for the query relation $\mathtt{r}$ with a policy network, and then feed these logic rules to a graph reasoning network, which applies the logic rules $\mathbf{z}$ to the background knowledge graph $\mathcal{G}$ for reasoning. In this way, the policy network defines a distribution $p_\theta(\boldsymbol{z}|\mathtt{r})$ and the reasoning network defines $p_\phi(\mathbf{y}|\mathcal{G}, \mathbf{x}, \mathbf{z})$, where θ and ϕ are parameters of the two networks respectively.

The target distribution can be factorized as follows:

$$p_{\phi,\theta}(\mathbf{y}|\mathcal{G},\mathbf{x}=(h,\mathtt{r},?))=\sum_{\mathbf{z}}p_{\phi}(\mathbf{y}|\mathcal{G},\mathbf{x},\mathbf{z})p_{\theta}(\mathbf{z}|\mathtt{r}). \tag{7.3.1}$$

The optimization problem for each training instance $(\mathcal{G},\mathbf{x},\mathbf{y})$ can be formulated as follows:

$$\max_{\phi,\theta}\log p_{\phi,\theta}(\mathbf{y}|\mathcal{G},\mathbf{x}=(h,\mathtt{r},?))=\log\sum_{\mathbf{z}}p_{\phi}(\mathbf{y}|\mathcal{G},\mathbf{x},\mathbf{z})p_{\theta}(\mathbf{z}|\mathtt{r}). \tag{7.3.2}$$

Next, we introduce how we design the policy network and the reasoning network.

7.3.1. Policy Networks

The goal of the policy network is to generate a set of useful logic rules for dealing with each query relation, and hence the network defines the distribution $p_\theta(\boldsymbol{y}|\mathtt{r})$ with $\mathtt{r}$ being the query relation.

As observed in existing works, chain-like rules play important roles in knowledge graph reasoning, and hence this paper focuses on learning chain-like rules. Each chain-like rule has the conjunctive form $\forall\{X_i\}_{i=0}^{l}\ \mathtt{r}(X_0,X_l)\leftarrow\mathtt{r}_1(X_0,X_1)\wedge\cdots\wedge\mathtt{r}_l(X_{l-1},X_l)$ and thus can be naturally represented as a sequence of relations $[\mathtt{r},\mathtt{r}_1,\mathtt{r}_2\cdots\mathtt{r}_l]$, where $\mathtt{r}$ is the head relation and $\{\mathtt{r}_i\}_{i=1}^{l}$ are body relations.

Intuitively, the chain-like rules can be generated by LSTM models [49], and thus we employ an LSTM-based rule generator. Given a query relation $\mathtt{r}$, the model generates body relation $[\mathtt{r}_1,\mathtt{r}_2\cdots\mathtt{r}_l]$ of chain-like rules in an autoregressive fashion. By doing this, the probability of each logic rule being generated can be simultaneously computed, and thus the rule generator defines a distribution over all chain-like rules. Suppose this distribution is $p_\theta^{\text{rule}}(\cdot|\mathtt{r})$, we further define the distribution over sets of logic rules as the following multinomial distribution:

$$p_\theta(\boldsymbol{y}|\mathtt{r})=\text{Mu}(\boldsymbol{z}|n_{\text{rule}},p_\theta^{\text{rule}}(\cdot|\mathtt{r})), \tag{7.3.3}$$

where Mu stands for the multinomial distribution, n_{rule} is a hyperparameter for the size of the set $\boldsymbol{z}$, and $p_\theta^{\text{rule}}(\cdot|\mathtt{r})$ defines a distribution over chain-like rules with head being $\mathtt{r}$. The generative process of a rule set $\hat{\boldsymbol{z}}$ is straightforward, where we simply generate n_{rule} rules with $p_\theta^{\text{rule}}(\cdot|\mathtt{r})$ to form $\hat{\boldsymbol{z}}$.

7.3.2. Reasoning Networks

The reasoning network defines likelihood of answer, i.e., $p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z})$. Intuitively, given a query $\boldsymbol{x}$, the reasoning predictor applies logic rules in $\boldsymbol{z}$ to the knowledge graph $\mathcal{G}$ for predicting the answer $\boldsymbol{y}$.

To do that, we notice that for each chain-like rule $\mathtt{r}\leftarrow\mathtt{r}_1\wedge\cdots\wedge\mathtt{r}_l$, if we start from the entity h in query $\boldsymbol{x}$ and sequentially walk along edges corresponding to body relations

$\mathtt{r}_1, \mathtt{r}_2, \cdots, \mathtt{r}_l$, we will arrive at a collection of destination entities. By doing that for all the logic rules in $\boldsymbol{z}$, we are able to obtain a bipartite factor graph, where one set of nodes correspond to chain-like rules in $\boldsymbol{z}$, and the other set of nodes are entities in the knowledge graph $\mathcal{G}$. Each edge of the bipartite graph links a logic rule f and an entity e, and the weight is defined as the number of relational paths starting from h, ending at e, and found by rule f.

With this bipartite graph, predicting the answer becomes a graph reasoning problem, i.e., identifying the answer entity in the bipartite graph. For this problem, we leverage an energy-based model, which computes an energy score $E_\phi(e)$ for each entity e in the bipartite graph, and thus the likelihood function is defined as follows:

$$p_\phi(\boldsymbol{y} = e|\mathcal{G}, \boldsymbol{x}, \boldsymbol{z}) = \frac{\exp(-E_\phi(e))}{\sum_{e' \in \mathcal{V}} \exp(-E_\phi(e'))}, \tag{7.3.4}$$

where $\mathcal{V}$ is the entity vocabulary of knowledge graph $\mathcal{G}$. For the energy $E_\phi(e)$ of each entity e, we consider information from two sources, which are the entity-wise information and the rule-wise information, corresponding to the information coming from entity e itself and from all logic rules in $\boldsymbol{z}$. With this idea, we have $E_\phi(e) = E_\phi^{\text{entity}}(e) + E_\phi^{\text{rule}}(e)$.

For the entity-wise energy $E_\phi^{\text{entity}}(e)$, we follow existing knowledge graph embedding models for parameterization. Specifically, a knowledge graph embedding model typically uses a score function $g(h, \mathtt{r}, e)$ to measure the inconsistency of the triplet $(h, \mathtt{r}, e)$, and thus lower scores imply that $(h, \mathtt{r}, e)$ is more likely a true fact. Based on that, we can naturally parameterize the entity-wise energy with the score function as $E_\phi^{\text{entity}}(e) = g(h, \mathtt{r}, e)$.

For the rule-wise energy $E_\phi^{\text{rule}}(e)$, we leverage a Transformer-based model, which computes the energy of an entity according to the logic rules connected with the entity. Formally, for each logic rule $f \in \boldsymbol{z}$, we can apply a Transforer encoder to the relation sequence of f, yielding a rule encoding as follows:

$$\boldsymbol{h}_f = \text{Transformer}_\phi(\mathtt{r}, \mathtt{r}_1, \cdots, \mathtt{r}_l) \quad \text{with} \quad f = \mathtt{r} \leftarrow \mathtt{r}_1 \wedge \cdots \wedge \mathtt{r}_l. \tag{7.3.5}$$

Once we have the encoding $\boldsymbol{h}_f$ of each logic rule, we can compute the rule-wise energy of each entity based on rule encodings. For each entity e, we consider logic rules connected to e, and use an aggregator to aggregate the encodings of these logic rules, yielding an entity encoding. Then we further apply an MLP, which projects the encoding to a rule-wise scalar energy as follows:

$$E_\phi^{\text{rule}}(e) = \text{MLP}_\phi(\text{Aggregator}(\{\boldsymbol{h}_f\}_{f \in \boldsymbol{z}})). \tag{7.3.6}$$

In practice, we use the PNA aggregator for its effectiveness. The pair-wise energy considers logic rules and does not consider any entity-specific information, which is thus generalizable.

With the above reasoning predictor, we are able to leverage both the entity-wise information and rule-wise information for reasoning.

7.3.3. End-to-end Optimization

Although the above policy network and reasoning network bring both interpretability and capacity to our approach, how to optimize them in an end-to-end fashion remains a challenging problem, as the set of logic rules $\boldsymbol{z}$ serves as a latent variable. In this paper, we resort to gradient ascent and importance sampling for a solution.

Specifically, for each training instance $(\mathcal{G}, \boldsymbol{x} = (h, \mathtt{r}, ?), \boldsymbol{y} = t)$, the gradient of the log-likelihood function with respect to θ and ϕ is given as follows:

$$\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G}, \boldsymbol{x}) = \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})], \tag{7.3.7}$$

$$\nabla_\phi \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G}, \boldsymbol{x}) = \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\phi \log p_\phi(\boldsymbol{y}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{z})]. \tag{7.3.8}$$

See Section E.1 for the detailed derivation. However, we see that the computation of gradient relies on the posterior distribution $p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y}) = p_\phi(\boldsymbol{y}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{z})p_\theta(\boldsymbol{z}|\mathtt{r})/Z_{\phi,\theta}$. The posterior distribution is intractable due to the partition function $Z_{\phi,\theta} = \sum_{\boldsymbol{z}} p_\phi(\boldsymbol{y}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{z})p_\theta(\boldsymbol{z}|\mathtt{r})$.

To address the challenge, we resort to importance sampling, where a proposal distribution $q(\boldsymbol{z})$ is introduce. The proposal distribution $q(\boldsymbol{z})$ can be any distribution which is easy to draw samples from and satisfies $q(\boldsymbol{z}) > 0$ for any set $\boldsymbol{z}$ of logic rules. We will discuss how to specify the proposal distribution in the next section. Suppose we have n samples (i.e., n sets of logic rules) drawn from $q(\boldsymbol{z})$ and denote the samples as $\hat{\boldsymbol{z}}_i$ for $i = \{1,2,\cdots,n\}$. The gradient with respect to θ can be estimated as follows:

$$\begin{aligned}
&\mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})] = \mathbb{E}_{q(\boldsymbol{z})}\left[\frac{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})}{q(\boldsymbol{z})}\nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})\right] \\
&\simeq \frac{1}{n}\sum_{i=1}^{n}\frac{p_{\phi,\theta}(\hat{\boldsymbol{z}}_i|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})}{q(\hat{\boldsymbol{z}}_i)}\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_i|\mathtt{r}) = \frac{1}{n}\sum_{i=1}^{n} r(\hat{\boldsymbol{z}}_i|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_i|\mathtt{r}),
\end{aligned} \tag{7.3.9}$$

where $r(\hat{\boldsymbol{z}}_i|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y}) = \frac{p_{\phi,\theta}(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}{q(\hat{\boldsymbol{z}}_i)} = \frac{p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z})p_\theta(\boldsymbol{z}|\mathtt{r})}{q(\hat{\boldsymbol{z}}_i)Z_{\phi,\theta}}$ is the ratio of the posterior and the proposal.

With importance sampling, we no longer need to sample from the posterior. However, we still need to compute the density of the posterior in the ratio $r(\hat{\boldsymbol{z}}_i|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})$, which is not desired due to the partition function $Z_{\phi,\theta}$. To avoid the problem, we notice the following approximation:

$$1 = \mathbb{E}_{q(\boldsymbol{z})}\left[\frac{p_\theta(\boldsymbol{z}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})}{q(\boldsymbol{z})}\right] = \mathbb{E}_{q(\boldsymbol{z})}[r(\boldsymbol{z}|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y})] \simeq \frac{1}{n}\sum_{i=1}^{n} r(\hat{\boldsymbol{z}}_i|\mathcal{G}, \boldsymbol{x}, \boldsymbol{y}), \tag{7.3.10}$$

which implies $\sum_{i=1}^{n} r(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y}) \simeq n$. By taking the approximation of n into the above equation, we obtain the following approximation:

$$\begin{aligned}\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) &\simeq \frac{1}{n}\sum_{i=1}^{n} r(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_i|\mathtt{r}) \\ &\approx \sum_{i=1}^{n} \frac{r(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}{\sum_{j=1}^{n} r(\hat{\boldsymbol{z}}_j|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_i|\mathtt{r}) \\ &= \sum_{i=1}^{n} \frac{\tilde{r}(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}{\sum_{j=1}^{n} \tilde{r}(\hat{\boldsymbol{z}}_j|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}\nabla_\theta \log p_\theta(\hat{\boldsymbol{z}}_i|\mathtt{r}),\end{aligned} \tag{7.3.11}$$

where $\tilde{r}(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y}) = \frac{p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z})p_\theta(\boldsymbol{z}|\mathtt{r})}{q(\hat{\boldsymbol{z}}_i)}$. Compared with $r(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})$, we can see that $\tilde{r}(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})$ is more tractable, as the partition function $Z_{\phi,\theta}$ is not needed. As a result, the above gradient can be easily computed, and intuitively the overall gradient is a weighted average of the gradient for each $\hat{\boldsymbol{z}}_i$.

Similarly, the gradient for the parameter ϕ of the reasoning predictor can be computed as below:

$$\nabla_\phi \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) \approx \sum_{i=1}^{n} \frac{\tilde{r}(\hat{\boldsymbol{z}}_i|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}{\sum_{j=1}^{n} \tilde{r}(\hat{\boldsymbol{z}}_j|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}\nabla_\phi \log p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\hat{\boldsymbol{z}}_i). \tag{7.3.12}$$

In this way, we are able to optimize the model in an end-to-end fashion. One key factor of the training algorithm lies on the choice of the proposal distribution. Next, we introduce how we choose the proposal distribution.

Choice of the Proposal Distribution. In the optimization algorithm, the proposal distribution probes different sets of logic rules for training the policy and the reasoning network. As the number of possible rule sets is huge, one natural idea for efficient search is to balance exploiting promising rule sets with exploring new rule sets. Towards this goal, we propose several different strategies for probing.

- **Random Exploration.** For this strategy, we mine logic rules from the observed facts using depth-first search. Then we randomly sample logic rules with equal probabilities as the proposal distribution. In this way, our approach is able to explore different logic rules.
- **Logic Rules Generated by Other Methods.** Another method we can use is to generate logic rules by using existing methods, e.g., RNNLogic. By doing this, our approach focuses more on exploiting rules viewed important by other methods, which is thus more efficient.
- **Self-exploration.** We can also use the policy network itself to generate logic rules, yielding a self-exploration method. With this method, the proposal distribution can be dynamically updated to balance between exploiting those useful logic rules and exploring those new logic rules.

Category	Algorithm	FB15k-237					WN18RR				
		MR	**MRR**	**H@1**	**H@3**	**H@10**	**MR**	**MRR**	**H@1**	**H@3**	**H@10**
Embedding	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
	ComplEx-N3*	-	0.37	-	-	56	-	0.48	-	-	57
	ConvE*	244	0.325	23.7	35.6	50.1	4187	0.43	40	44	52
	TuckER*	-	0.358	26.6	39.4	54.4	-	0.470	44.3	48.2	52.6
	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	-	-	-	-	-
	M-Walk*	-	0.232	16.5	24.3	-	-	0.437	41.4	44.5	-
	RNNLogic*	232	0.344	25.2	38.0	53.0	4615	0.483	44.6	49.7	55.8
	NBFNet*	**114**	**0.415**	**32.1**	**45.4**	**59.9**	**636**	**0.551**	**49.7**	**57.3**	**66.6**
DiffLogic	only rule	296	0.346	25.8	37.8	51.9	5796	0.522	48.1	54.4	60.4
	full	154	0.375	28.2	41.1	56.1	4457	0.525	48.3	54.5	60.9

Table 31. Results of DiffLogic on FB15k-237 and WN18RR.

Category	Algorithm	Kinship					UMLS				
		MR	**MRR**	**H@1**	**H@3**	**H@10**	**MR**	**MRR**	**H@1**	**H@3**	**H@10**
Embedding	DistMult	8.5	0.354	18.9	40.0	75.5	14.6	0.391	25.6	44.5	66.9
	ComplEx	7.8	0.418	24.2	49.9	81.2	13.6	0.411	27.3	46.8	70.0
	ComplEx-N3	-	0.605	43.7	71.0	92.1	-	0.791	68.9	87.3	95.7
	TuckER	6.2	0.603	46.2	69.8	86.3	5.7	0.732	62.5	81.2	90.9
	RotatE	3.7	0.651	50.4	75.5	93.2	4.0	0.744	63.6	82.2	93.9
Rule Learning	MLN	10.0	0.351	18.9	40.8	70.7	7.6	0.688	58.7	75.5	86.9
	Boosted RDN	25.2	0.469	39.5	52.0	56.7	54.8	0.227	14.7	25.6	37.6
	PathRank	-	0.369	27.2	41.6	67.3	-	0.197	14.8	21.4	25.2
	NeuralLP	16.9	0.302	16.7	33.9	59.6	10.3	0.483	33.2	56.3	77.5
	DRUM	11.6	0.334	18.3	37.8	67.5	8.4	0.548	35.8	69.9	85.4
	MINERVA	-	0.401	23.5	46.7	76.6	-	0.564	42.6	65.8	81.4
	CTP	-	0.335	17.7	37.6	70.3	-	0.404	28.8	43.0	67.4
	RNNLogic	3.1	**0.722**	**59.8**	**81.4**	94.9	3.1	0.842	77.2	89.1	96.5
DiffLogic	only rule	3.4	0.648	49.6	75.0	94.5	3.2	0.841	77.2	89.6	95.8
	full	**3.0**	0.718	58.9	81.3	**95.5**	**3.0**	**0.863**	**80.4**	**90.6**	**96.6**

Table 32. Results of DiffLogic on Kinship and UMLS.

7.4. Experiment

7.4.1. Settings

Datasets. We consider four commonly-used datasets for evaluation, including including FB15k-237 [132], WN18RR [26], Kinship, and UMLS [63]. For Kinship and UMLS, there exist several train/validation/test data splits, and we choose the one used in Qu et al. [105]

as they use fewer facts for training, which is more challenging. We summarize the detailed statistics of these datasets in Section E.2.

Compared Algorithms. The following algorithms are compared in experiment.

• *Statistical Relational Learning.* Some classical methods in the statistical relational learning literature can be applied to our problem, including Markov logic networks [106] and boosted relational dependency networks [88].

• *Path-based Methods.* Path-based methods indirectly learn logic rules by post-processing relational paths. For classical path-based methods, we compare against path ranking [71]. For methods based on neural logic programming and neural theorem provers, we consider NeuralLP [157], DRUM [108], NLIL [158], and CTP [84]. We also consider reinforcement learning methods, including MINERVA [23] and M-Walk [117]. Besides, we also consider NBFNet [171], which is a GNN-based method and achieves state-of-the-art results on knowledge graph reasoning.

• *Embedding Methods.* We also consider some embedding methods, including TransE [8], DistMult [156], ComplEx [135], ComplEx-N3 [68], ConvE [26], TuckER [4], and RotatE [121].

• *Our Approach.* For our approach, we consider two variants. The first variant only considers the rule-wise energy in the reasoning predictor, and we denote the model as *only rule*. The second variant uses both rule-wise and entity-wise energy in reasoning networks, and the model is denoted as *full*.

Evaluation Metrics. For each fact $(h, \mathtt{r}, t)$ in the test set, we consider two queries $(h, \mathtt{r}, ?)$ and $(t, \mathtt{r}^{-1}, ?)$ with answers t and h. For each query, we compute a score for each entity in the entity set and further consider the rank of the answer. Then the Mean Rank (MR), Mean Reciprocal Rank (MRR), and Hit@k over all queries are reported. We always consider the filtered setting [8] as it is used in most studies. Besides, there can be cases where multiple entities receive the same score in a query. In such cases, we follow Qu et al. [105] to compute the expectation of each metric over all random shuffles of entities receiving the same score.

Experimental Setup. In our problem, we observe a collection of facts in the training set. For each of this $(h, \mathtt{r}, t)$ fact, we add its inverse fact $(t, \mathtt{r}^{-1}, h)$ into the training fact set. To build a training instance, we randomly sample a training fact $(h, \mathtt{r}, t)$. Then we form the query and answer as $\boldsymbol{x} = (h, \mathtt{r}, ?)$ and $\boldsymbol{a} = t$. The background knowledge graph $\mathcal{G}$ is formted as the rest of training facts. During testing, the background knowledge graph $\mathcal{G}$ is formed with all the triplets in $\mathcal{T}$. In the policy network for rule generation, the maximum length of logic rules is set to 3 for FB15k-237, 5 for WN18RR, and 3 for Kinship and UMLS based on validation performance. For the proposal distribution, we sample 100 rules for each length to form the rule set $\boldsymbol{z}$.

7.4.2. Results

We present the main results in Table 31 and Table 32, where H@k is in %. [*] means the numbers are taken from the original papers. [†] means we rerun the methods with the same evaluation process. For our approach which only uses rule-wise energy in the reasoning network (e.g., *only rule* model), it outperforms state-of-the-art knowledge graph embedding models such as RotatE and TuckER in most cases, which shows that these logic rules are helpful in knowledge graph reasoning. Besides, this variant also outperforms most well-known methods for logic rule learning, including NeuralLP, M-Walk, and RNNLogic. The performance gain mainly comes from the use of the graph reasoning network.

By further considering the entity-wise energy in the reasoning network (e.g., *full* model), our approach further achieves significant improvement, especially on FB15k-237 and Kinship. Moreover, the result of this variant is comparable to NBFNet, which is the state-of-the-art approach, but our approach has better interpretability, as it explicitly generates many logic rules which are useful for reasoning.

Chapter 8

Application

In the preceding chapters, we introduced our innovative paradigm, which combines deep learning with statistical relational learning to facilitate reasoning on graph-based data. Throughout these chapters, we provided several illustrative examples showcasing the versatility of this paradigm across various graph reasoning tasks.

While our primary focus in the aforementioned chapters was on the domain of graph machine learning, graph-structured data also finds relevance in other fields, such as natural language processing and computer vision. Thus, our proposed paradigm holds the potential to offer significant benefits when applied to applications within these domains.

Indeed, this thesis not only delves into the application of our paradigm in other domains but also presents concrete examples to demonstrate its efficacy. In this chapter, we offer several practical instances of its utilization, including scene graph generation, learning on text-attributed graphs, and few-shot sentence classification.

8.1. Scene Graph Generation

In Xu et al. [154] where I involve as the second author, we apply our proposed paradigm to scene graph generation, which is a crucial problem in computer vision. Specifically, we integrate object instance segmentation models [47] in the deep learning field with the conditional random field (CRF) [69] in statistical relational learning.

8.1.1. Problem Definition

The goal of the problem is to extract a *scene graph*, i.e., a structured representation of visual scene from an image. Formally, we define a scene graph as $G = (y_O, R)$. y_O denotes the category labels of the objects O in the image, and it holds that $y_o \in \mathcal{C}$ for each object $o \in O$, where $\mathcal{C}$ stands for the set of all object categories, including the "background" category. $R = \{(o_h,r,o_t)\}$ is the set of relational triplets/edges with $r \in \mathcal{T}$ as the relation type from

head object o_h to tail object o_t ($o_h, o_t \in O$), where $\mathcal{T}$ represents all relation types, including the type of "no relation". With the above information, we aim at jointly modeling visual objects and visual relations as defined below:

Joint Scene Graph Modeling. Given an image I, we aim to jointly predict object categories y_O and the relationships R among all objects, which models the joint distribution of scene graphs, i.e., $p(G|I) = p(y_O, R|I)$.

8.1.2. Model

To address this problem, we introduce Joint Modeling for Scene Graph Generation (JM-SGG). Existing approaches typically tackle this challenge by independently predicting object and relation labels, relying on informative representations. However, this approach limits the extent to which different labels can benefit from the predictions of each other. JM-SGG overcomes this limitation by employing a unified conditional random field (CRF) that jointly models all objects and relationships within a visual scene. This unified approach enables various object and relation labels to interact more effectively. Nevertheless, mastering the intricacies of this complex CRF poses a nontrivial challenge. To address this, we propose a solution that combines maximum likelihood estimation with mean-field variational inference. The variational distribution is parameterized using deep learning models, resulting in an efficient algorithm for both learning and inference.

In essence, our method leverages the CRF of the statistical relational learning field to capture dependencies among object labels and relationships, while harnessing the power of deep learning models to enhance the inference and learning processes.

Probabilistic Formalization. In the JM-SGG model, we organize the observed scene image I and all object and relation labels in the latent scene graph (i.e., y_O and R) as the nodes in a unified conditional random field. Since the interactions of these nodes are either for a single object or for the relationship between an object pair, we decompose the graphical structure of whole network into two sets of components. (1) *Object components*: For an object $o \in O$, we consider the dependency of its category label on its visual representation and thus connect y_o with I. (2) *Relation components*: for a relational triplet (o_h,r,o_t) $\in R$, we consider the dependency of relation type r on the visual cues in image I, and we also model the interdependency among the object and relation labels in this triplet (i.e., y_{o_h}, y_{o_t} and r), which forms a relation component. By combining all object and relation components, the CRF can capture the comprehensive label dependency within a scene graph. We now define the joint distribution of scene graphs upon the observed scene image as below:

$$p_\Theta(G|I) = \frac{1}{Z_\Theta(I)} \, f_\Theta(G, I), \tag{8.1.1}$$

$$f_\Theta(G, I) = \prod_{o \in O} \phi(y_o, I) \prod_{(o_h, r, o_t) \in R} \psi(r, y_{o_h}, y_{o_t}, I), \tag{8.1.2}$$

where Θ summarizes the parameters of whole model, f_Θ is an unnormalized likelihood function, Z_Θ denotes the partition function, and ϕ and ψ are the potential functions defined on object and relation components respectively. In this thesis, we omit the details on how we parameterize the potential functions, so that audience can focus on the high-level idea of how we integrate deep learning and statistical relational learning.

Learning. In the learning phase, we seek to learn the parameters of potential functions by maximum likelihood estimation, where Θ summarizes all these parameters. Specifically, we aim to maximize the expectation of log-likelihood function $\log p_\Theta(G|I)$ with respect to the data distribution p_d, *i.e.* $\mathcal{L}(\Theta) = \mathbb{E}_{G \sim p_d}\Big[\log p_\Theta(G|I)\Big]$. However, the exact computation of $\mathcal{L}(\Theta)$ is impeded by the intractable partition function $Z_\Theta(I)$ which sums over all possible scene graphs. Therefore, we resort to gradient ascent, in which the gradient of the objective function $\mathcal{L}(\Theta)$ with respect to Θ can be computed as below:

$$\nabla_\Theta \mathcal{L}(\Theta) = \mathbb{E}_{G \sim p_d}[\nabla_\Theta \log f_\Theta(G, I)] - \mathbb{E}_{G \sim p_\Theta}[\nabla_\Theta \log f_\Theta(G, I)], \tag{8.1.3}$$

where p_Θ is the model distribution that approximates p_d (i.e., the posterior distribution $p_\Theta(G|I)$ defined by JM-SGG model). In practice, we estimate the first expectation in the above equation with the ground-truth scene graphs in a mini-batch. The estimation of the second expectation is nontrivial, which requires to sample scene graphs from the intractable model distribution. One option is to run the Markov Chain Monte Carlo (MCMC) sampler, but its computational cost is high, and we therefore use mean-field variational inference for more efficient sampling as introduced next.

Inference. The inference phase aims to compute the posterior distribution $p_\Theta(G|I)$ and also sample from it. Exact inference is always infeasible due to the complex structure among the latent variables y_O and R of the scene graph. Therefore, we approximate $p_\Theta(G|I)$ with a variational distribution $q_\Omega(G)$ via the mean-field approximation:

$$q_\Omega(G) = \prod_{o \in O} q_\Omega(y_o) \prod_{(o_h, r, o_t) \in R} q_\Omega(r), \tag{8.1.4}$$

where each factor $q_\Omega(y_o)$ and $q_\Omega(r)$ defines a categorical distribution. Similar to models introduced in previous chapters, these factors are parameterized by deep learning models, which significantly improves the effectiveness and efficiency of inference. Due to the limited space, we omit the details.

8.1.3. Article Details

This section includes materials from the following paper where I get involved as the second author. My contribution is summarized as follows:

- **Joint Modeling of Visual Objects and Relations for Scene Graph Generation.** Minghao Xu, Meng Qu, Bingbing Ni, Jian Tang. *Advances in Neural Information Processing Systems, 2021.*

  Personal Contribution. I participated in conceiving the initial idea, formulating the mathematical foundations of the approach, and contributed to crafting specific sections of the paper. Minghao also contributed to the conceptualization of the idea, collaborated on mathematical formulation, took charge of model implementation, conducted all experiments, and contributed to the content of the paper. Bingbing played a role in shaping the idea and contributed to refining the paper writing. Jian was instrumental in brainstorming the idea, providing valuable insights into its development, and played a pivotal role in overseeing the project progression while also contributing to the final refinement.

8.2. Learning on Text-attributed Graphs

In Zhao et al. [167] which I contribute as the co-first author, we leverage our paradigm for learning on text-attributed graphs, which is an important problem in both natural language processing and graph machine learning. For this problem, we integrate relational Markov networks (RMN) [127] in statistical relational learning and pre-trained language models (PLM) [55] in deep learning.

8.2.1. Problem Definition

A Text-Attributed Graph (TAG) $\mathcal{G}_S = (V, A, \mathbf{s}_V)$ is composed of nodes V and their adjacency matrix $A \in \mathbb{R}^{|V|\times|V|}$, where each node $n \in V$ is associated with a sequential text feature (sentence) $\mathbf{s}_n$. There are a variety of reasoning tasks on a TAG, and here we take node classification as an example to study as it is the most important task on TAGs. Given a few labeled nodes $\mathbf{y}_L$ of $L \subset V$, the goal is to predict the labels $\mathbf{y}_U$ for the remaining unlabeled objects $U = V \setminus L$.

8.2.2. Model

We present GLEM as a solution to address the previously described problem. In a manner akin to GMNN, GLEM defines the joint distribution of node labels using a relational Markov network (RMN), effectively capturing the intricate dependencies among node labels. However, inference and learning within the RMN framework remain a challenge. To tackle this, GLEM employs mean-field inference and parameterizes the variational distribution with pre-trained language models (PLM), such as BERT. These two crucial components, RMN and PLM, undergo an alternating optimization process within an EM framework.

To summarize, the RMN in GLEM adeptly models label dependencies among nodes, enhancing the performance of the PLM. Simultaneously, the PLM substantially boosts the efficiency and effectiveness of both RMN inference and learning, resulting in a harmonious fusion of these components for node classification in TAGs.

Probabilistic Formalization. GLEM is based on a pseudo-likelihood variational framework, which offers a principled and flexible formalization for model design. The framework tries to maximize the log-likelihood function of the observed node labels, i.e., $p(\mathbf{y}_L|\mathbf{s}_V, A)$. Directly optimizing the function is often hard due to the unobserved node labels $\mathbf{y}_U$, and thus the framework instead optimizes the evidence lower bound as below:

$$\log p(\mathbf{y}_L|\mathbf{s}_V, A) \geq \mathbb{E}_{q(\mathbf{y}_U|\mathbf{s}_U)}[\log p(\mathbf{y}_L, \mathbf{y}_U|\mathbf{s}_V, A) - \log q(\mathbf{y}_U|\mathbf{s}_U)], \tag{8.2.1}$$

where $q(\mathbf{y}_U|\mathbf{s}_U)$ is a variational distribution and the above inequality holds for any q. The ELBO can be optimized by alternating between optimizing the distribution q (i.e., E-step) and the distribution p (i.e., M-step). In the E-step, we aim at updating q to minimize the KL divergence between $q(\mathbf{y}_U|\mathbf{s}_U)$ and $p(\mathbf{y}_U|\mathbf{s}_V, A, \mathbf{y}_L)$, so that the above lower bound can be tightened. In the M-step, we then update p towards maximizing the following pseudo-likelihood function:

$$\mathbb{E}_{q(\mathbf{y}_U|\mathbf{s}_U)}[\log p(\mathbf{y}_L, \mathbf{y}_U|\mathbf{s}_V, A)] \approx \mathbb{E}_{q(\mathbf{y}_U|\mathbf{s}_U)}[\sum_{n\in V} \log p(\mathbf{y}_n|\mathbf{s}_V, A, \mathbf{y}_{V\setminus n})]. \tag{8.2.2}$$

Next, we introduce how we apply the framework to node classification in TAGs by instantiating the p and q distributions with RMNs and PLMs respectively.

Parameterization. The distribution q aims to use the text information $\mathbf{s}_U$ to define node label distribution. In GLEM, we use a mean-field form, assuming the labels of different nodes are independent and the label of each node only depends on its own text information, yielding the following form of factorization:

$$q_\theta(\mathbf{y}_U|\mathbf{s}_U) = \prod_{n\in U} q_\theta(\mathbf{y}_n|\mathbf{s}_n). \tag{8.2.3}$$

Each term $q_\theta(\mathbf{y}_n|\mathbf{s}_n)$ can be modeled by a PLM q_θ parameterized by θ, which effectively models the fine-grained token interactions.

On the other hand, the distribution p defines a joint distribution $p(\mathbf{y}_V|\mathbf{s}_V, A)$, aiming to model the dependency of node labels. Thus, we apply a RMN p_ϕ to parameterize the joint distribution with ϕ being the model parameters. During both inference and learning, only the joint distribution $p_\phi(\mathbf{y}_n|\mathbf{s}_V, A, \mathbf{y}_{\mathrm{NB}(n)})$ induced by the RMN p_ϕ is required, so we follow GMNN and characterize the RMN with a graph neural network.

Optimization. The optimization process of GLEM remains very similar to the process of GMNN, where an EM framework is employed. Due to the limited space, we omit the details

of the optimization algorithm. For audience who are interested in the optimization process, please refer to Algorithm 1.

8.2.3. Article Details

The materials of this section come from the following paper where I contribute as a co-first author. The details of the paper and my personal contribution are summarized as below (* stands for equal contribution):

- **Learning on Large-scale Text-attributed Graphs via Variational Inference.** Jianan Zhao*, Meng Qu*, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, Jian Tang. *International Conference on Learning Representations, 2023.*
  Personal Contribution. I involved in the inception of the initial idea, contributed to the development of mathematical formulations, played an active role in designing experiments, and wrote the majority of the paper. Jianan was also instrumental in the conception of the initial idea, took charge of model implementation, fine-tuned its performance, conducted a significant portion of the experiments, and initiated the first draft of the paper. Hao played a vital role in refining the model performance and actively contributed to the experimental phase. Chaozhuo, Qian, Rui, and Xing made valuable contributions through extensive discussions on the idea and participated in the collaborative paper writing process. Jian contributed to the formulation of the idea, enhanced the quality of paper writing, and provided valuable project supervision.

8.3. Few-shot Sentence Classification

In Qu et al. [104] where I was the first author, the paradigm proposed in this thesis is applied to few-shot relation extraction, which is essentially a few-shot sentence classification problem and has a variety of applications in the field of natural language processing. For this problem, we leverage a probabilistic formalization inspired by statistical relational learning, and meanwhile apply the pre-trained language models (PLM) [55] in deep learning to facilitate the inference process.

8.3.1. Problem Definition

Relation extraction is an important task in many research areas, which aims at predicting the relation of two entities given a sentence. Existing methods typically require a large number of labeled sentences for training, which are expensive to obtain. Thus, recent studies focus on few-shot relation extraction, where only few examples for each relation are given as training data. However, the results are still far from satisfactory due to limited information in the few examples. To further improve the results, another data source should be considered.

Therefore, we propose to study few-shot relation extraction with a global graph of relations, where a global graph describing the connections of all possible relations is assumed to be an additional data source. More formally, we denote the global relation graph as $\mathcal{G} = \{\mathcal{R}, \mathcal{L}\}$, where $\mathcal{R}$ is the set of all the possible relations, and $\mathcal{L}$ is a collection of links between different relations. The linked relations are likely to have similar semantic meanings.

In few-shot relation extraction, each time we only consider a subset of relations from the whole relation set, i.e., $\mathcal{T} \subseteq \mathcal{R}$. Given a few support sentences $\mathcal{S}$ of these relations, where $\mathbf{x}_\mathcal{S} = \{\mathbf{x}_s\}_{s\in\mathcal{S}}$ represents the text of these sentences, and $\mathbf{y}_\mathcal{S} = \{\mathbf{y}_s\}_{s\in\mathcal{S}}$ represents the corresponding labels with each $\mathbf{y}_s \in \mathcal{T}$, our goal is to learn a classifier for these relations using the global graph and support sentences. Then for some unlabeled query sentences $\mathbf{x}_\mathcal{Q} = \{\mathbf{x}_q\}_{q\in\mathcal{Q}}$, we apply the classifier to predict their labels $\mathbf{y}_\mathcal{Q} = \{\mathbf{y}_q\}_{q\in\mathcal{Q}}$ with each $\mathbf{y}_q \in \mathcal{T}$.

8.3.2. Model

We introduce REGRAB as a solution to the problem at hand. Drawing inspiration from established methods in statistical relational learning, we reframe the problem from a probabilistic perspective. This approach offers a deeper insight into the problem, allowing us to effectively capture the interdependencies among support sentences and query sentences. To enhance the inference process of our model, we take advantage of a pre-trained language model (PLM) to improve both effectiveness and efficiency.

In summary, REGRAB borrows insights from the field of statistical relational learning to formalize the problem and model the intricate relationships among support sentences and query sentences, thereby enhancing the predictive capabilities of the PLM. Concurrently, the incorporation of the PLM significantly improves the efficiency and effectiveness of the inference process in REGRAB.

Probabilistic Formalization. REGRAB gets inspiration from statistical relational learning and formalizes the problem in a probabilistic way. More specifically, recall that given a subset of relations $\mathcal{T} \subseteq \mathcal{R}$, the goal is to predict the labels $\mathbf{y}_Q$ of some query texts $\mathbf{x}_Q$ based on a global task graph $\mathcal{G}$ and few support sentences $(\mathbf{x}_S, \mathbf{y}_S)$. Formally, our goal can be stated as computing the following log-probability:

$$\log p(\mathbf{y}_Q|\mathbf{x}_Q, \mathbf{x}_S, \mathbf{y}_S, \mathcal{G}). \tag{8.3.1}$$

We compute the probability by representing each relation $r \in \mathcal{T}$ with a prototype vector $\mathbf{v}_r$, which summarizes the semantic meaning of that relation. By introducing such prototype vectors, the log-probability can be factorized as:

$$\log p(\mathbf{y}_Q|\mathbf{x}_Q, \mathbf{x}_S, \mathbf{y}_S, \mathcal{G}) = \log \int p(\mathbf{y}_Q|\mathbf{x}_Q, \mathbf{v}_\mathcal{T}) p(\mathbf{v}_\mathcal{T}|\mathbf{x}_S, \mathbf{y}_S, \mathcal{G}) d\mathbf{v}_\mathcal{T} \tag{8.3.2}$$

where $\mathbf{v}_{\mathcal{T}} = \{\mathbf{v}_r\}_{r\in\mathcal{T}}$ is a collection of prototype vectors for all the target relations in $\mathcal{T}$. These prototype vectors are characterized by both the labeled sentences in the support set and global task graph through the distribution $p(\mathbf{v}_{\mathcal{T}}|\mathbf{x}_S, \mathbf{y}_S, \mathcal{G})$. With such prototype vectors to represent target relations, the distribution of query sentence labels can then be defined through a softmax function as follows:

$$\begin{aligned} p(\mathbf{y}_Q|\mathbf{x}_Q, \mathbf{v}_{\mathcal{T}}) &= \prod_{q\in Q} p(\mathbf{y}_q|\mathbf{x}_q, \mathbf{v}_{\mathcal{T}}), \text{ with each} \\ p(\mathbf{y}_q = r|\mathbf{x}_q, \mathbf{v}_{\mathcal{T}}) &= \frac{\exp(\mathcal{E}(\mathbf{x}_q)\cdot\mathbf{v}_r)}{\sum_{r'\in\mathcal{T}}\exp(\mathcal{E}(\mathbf{x}_q)\cdot\mathbf{v}_{r'})}, \end{aligned} \tag{8.3.3}$$

where $\mathcal{E}$ is a sentence encoder, which encodes a query sentence $\mathbf{x}_q$ into an encoding $\mathcal{E}(\mathbf{x}_q)$. Intuitively, we compare the encoding with the prototype vector $\mathbf{v}_r$ of each relation to estimate how likely the sentence expresses the relation.

Under such a formalization, the key is how to parameterize $p(\mathbf{v}_{\mathcal{T}}|\mathbf{x}_S, \mathbf{y}_S, \mathcal{G})$, which is the posterior distribution of prototype vectors conditioned on the support sentences and the global relation graph. Next, we introduce how we parameterize the posterior distribution.

Parameterization of the Posterior Distribution. To model the posterior distribution of prototype vectors, we notice that the posterior can be naturally factorized into a prior distribution conditioned on the relation graph, and a likelihood function on the few support sentences. Therefore, we can formally represent the posterior as follows:

$$p(\mathbf{v}_{\mathcal{T}}|\mathbf{x}_S, \mathbf{y}_S, \mathcal{G}) \propto p(\mathbf{y}_S|\mathbf{x}_S, \mathbf{v}_{\mathcal{T}})p(\mathbf{v}_{\mathcal{T}}|\mathcal{G}), \tag{8.3.4}$$

where $p(\mathbf{v}_{\mathcal{T}}|\mathcal{G})$ is the prior for the prototype vectors and $p(\mathbf{y}_S|\mathbf{x}_S, \mathbf{v}_{\mathcal{T}})$ is the likelihood on support sentences. To effectively extract knowledge from the global task graph to characterize the prior distribution, we introduce a graph neural network. As this part is not our main focus, we will skip the details.

Besides the graph-based prior, we also consider the likelihood on support sentences when parameterizing the posterior distribution of prototype vectors. Similar to the likelihood on the query sentences, the likelihood on support sentences can be characterized as below:

$$\begin{aligned} p(\mathbf{y}_S|\mathbf{x}_S, \mathbf{v}_{\mathcal{T}}) &= \prod_{s\in S} p(\mathbf{y}_s|\mathbf{x}_s, \mathbf{v}_{\mathcal{T}}), \text{ with each} \\ p(\mathbf{y}_s = r|\mathbf{x}_s, \mathbf{v}_{\mathcal{T}}) &= \frac{\exp(\mathcal{E}(\mathbf{x}_s)\cdot\mathbf{v}_r)}{\sum_{r'\in\mathcal{T}}\exp(\mathcal{E}(\mathbf{x}_s)\cdot\mathbf{v}_{r'})}, \end{aligned} \tag{8.3.5}$$

where $\mathcal{E}$ is a sentence encoder. To improve the performance of inferring sentence labels, we parameterize the sentence encoder $\mathcal{E}$ with a PLM (e.g., BERT [55]).

Summary. Through a formalization inspired by statistical relational learning and the strategic utilization of pre-trained language models (PLMs) to enhance inference, REGRAB

adeptly captures the intricate dependencies between support sentences and query sentences. Consequently, this approach leads to precise inference of sentence labels.

Due to limited space, here we omit the details of how to optimize REGRAB. For audience interested in the optimization algorithm, please refer to the paper [104].

8.3.3. Article Details

This section mainly reuses the content of the following paper where I am the first author. We summarize the details and my contribution as follows:

- **Few-shot Relation Extraction via Bayesian Meta-learning on Task Graphs.** Meng Qu, Tianyu Gao, Louis-Pascal Xhonneux, Jian Tang. *International Conference on Machine Learning, 2020.*

  Personal Contribution. I conceived the idea, formulated the mathematical framework, implemented the model, conducted the majority of the experiments, and wrote most sections of the paper. Tianyu contributed to conducting specific experiments. Louis-Pascal played an engaged role in project discussions, offering valuable insights and perspectives. Jian was instrumental in shaping the initial idea through collaborative discussions, enhanced the quality of paper writing, and provided essential project supervision.

Chapter 9

Conclusion

This thesis primarily revolves around reasoning on graph-structured data, encompassing a spectrum of applications such as node classification, link prediction, and logic rule induction.

These tasks find solutions in two distinct yet harmonious realms: deep learning and statistical relational learning. These methodologies naturally complement each other. Statistical relational learning excels at modeling intricate dependencies among evidence and incorporates logic rules effectively. In contrast, deep learning methods excel at learning meaningful representations of objects, offering efficiency and high-performance capabilities across a diverse array of tasks. Recognizing the inherent synergy between these approaches, we introduce an innovative paradigm that unites them. Specifically, deep learning techniques play a pivotal role in speeding up the typically slower processes of inference and learning within statistical relational learning. Simultaneously, statistical relational learning methods refine the predictions generated by deep learning models, capitalizing on joint dependencies and logical rules. This integrated paradigm enables the development of efficient approaches capable of capturing complex logical dependencies.

In Section 3 and Section 4, we delve into node classification within the transductive and inductive settings respectively, introducing GMNN and SPN as our proposed solutions. These approaches seamlessly merge the principles of relational Markov networks and conditional random fields from the statistical relational learning domain with the power of graph neural networks from deep learning. This integration equips GMNN and SPN to learn meaningful node representations for accurate node label predictions while effectively capturing dependencies among node labels.

In Section 5, we tackle a more intricate challenge: knowledge graph reasoning, which is a task of predicting missing links within a knowledge graph. To address this, we introduce pLogicNet, a solution that marries knowledge graph embedding techniques from deep

learning with Markov logic networks from statistical relational learning. Through this fusion, pLogicNet harnesses pre-defined logic rules to enhance entity and relation embeddings, thereby advancing knowledge graph reasoning.

Section 6 and Section 7 further explore the automatic learning of logic rules, eliminating the need for manual rule specification. Toward this goal, we introduce RNNLogic and DiffLogic. These models integrate stochastic logic programming from the statistical relational learning realm for reasoning tasks (i.e., reasoning predictor) and employ deep generative models from deep learning for logic rule generation (i.e., rule generator). We present both an EM-based optimization algorithm and an end-to-end differentiable optimization algorithm, allowing effective joint optimization of the reasoning model and rule generator. This integration empowers us to automatically uncover crucial logic rules and achieve remarkable results in the task of knowledge graph reasoning.

The preceding chapters have applied our paradigm to pivotal applications within the field of graph machine learning. Importantly, this paradigm exhibits the potential to extend its utility to various other domains. In Section 8, we showcase three illustrative examples: scene graph generation, learning on text-attributed graphs, and few-shot sentence classification. These instances serve as compelling evidence of the versatility and adaptability of our proposed paradigm, highlighting its capacity to excel across a wide spectrum of applications.

9.1. Future Directions

Our paradigm of integrating deep learning and statistical relational learning opens up a vast array of promising avenues for future exploration and advancement.

9.1.1. Application to Large Language Models

In recent developments within the deep learning arena, large language models (LLMs) [93, 94, 133, 134] have demonstrated remarkable expressive capabilities in reasoning, yielding impressive outcomes across a diverse range of tasks.

Large language models tackle reasoning problems by framing them as a text generation task, with the ability to autonomously generate answers in an autoregressive manner. While this approach has proven effective, it lacks the structured reasoning process that humans employ, where we construct schema graphs in our minds to facilitate complex problem-solving. Some recent efforts aim to address this discrepancy by introducing various schema graphs into large language models to enhance their reasoning capabilities. Examples include chain-of-thought (CoT) [147] and tree-of-thought (ToT) [160], which introduce chain-like and tree-like structures, respectively, during the reasoning process of large language models.

Despite their promising results, these methods remain in relatively nascent stages compared to the structured way human reasoning operates. We firmly believe that our proposed paradigm offers a more principled solution to this challenge by combining large language models with statistical relational learning. By leveraging statistical relational learning, our paradigm can extract inherent logic and dependencies from observed data, upon which we can construct a schema graph and further build a probabilistic graphical model (e.g., Markov logic networks [106]) to aid large language models in their reasoning. This approach enables language models to engage in more structured reasoning processes, potentially leading to even more robust and refined results.

9.1.2. Application in Drug Discovery

The field of drug discovery has assumed paramount importance, especially in the wake of the COVID-19 pandemic, as it strives not only to combat infectious diseases but also to tackle severe conditions such as cancer, making significant strides for the benefit of humanity. Concurrently, the rapid advancement of geometric deep learning [9] has positioned these techniques as potent tools in the realm of drug discovery. The underlying concept involves constructing a graph that represents the spatial relationships among atoms in three-dimensional space and harnessing equivariant graph neural networks [113] to facilitate message passing among these atoms. These methods have delivered remarkable results across a diverse spectrum of tasks. Notably, based on the foundations of geometric deep learning, AlphaFold2 [52] and RosettaFold [2] have achieved groundbreaking outcomes in protein structure prediction, marking a significant milestone in computational biology and advancing the quest for antibody drug design.

While geometric deep learning has demonstrated impressive performance, it relies on a substantial volume of data to effectively learn the intricate patterns governing the interactions among various biomedical entities, including proteins, drugs, and antigens. However, the availability of such data in the realm of drug discovery remains constrained. This limitation impedes the ability of these models to acquire a comprehensive understanding of the mechanisms or rules governing entity interactions, occasionally resulting in deviations from established biomedical principles.

To address this challenge, a promising avenue involves applying our paradigm to integrate geometric deep learning and statistical relational learning. This approach enables us to explicitly leverage statistical relational learning methods to model the rules and insights distilled by human experts. These learned rules can then be employed to refine or regularize the predictions generated by geometric deep learning models, ultimately yielding enhanced prediction results and aligning more closely with established principles in biomedicine.

9.1.3. Alignment with the Consciousness Prior

The consciousness prior, as proposed by Bengio [5], introduces an innovative framework for learning high-level abstract representations. It conceptualizes consciousness as an attention mechanism that selects a limited set of concepts, condensing them into a low-dimensional conscious state. This conscious state serves as a critical bottleneck significantly influencing subsequent processing, with a direct correspondence to thoughts that can be articulated in natural language. Notably, the prior posits a joint distribution between high-level concepts characterized by a sparse factor graph, where each factor relates only to a small subset of concepts. This design aligns seamlessly with natural language and symbolic knowledge representations. The consciousness prior effectively facilitates the learning of disentangled representations and fosters systematic generalization, establishing connections between deep learning and higher cognitive functions such as reasoning, planning, and imagination.

The consciousness prior shares several key intuitions with our paradigm, particularly in their shared objective of integrating deep learning techniques with the cognitive capabilities akin to System 2 thinking. Both approaches seek to empower models with enhanced capacity for effective and intricate reasoning. Furthermore, both methods employ graph structures as abstract representations of high-level knowledge, leveraging them to enhance deep learning models. While the consciousness prior endeavors to unveil a sparse factor graph from observed evidence, our paradigm strives to learn logical patterns that elucidate the dependencies between objects, subsequently constructing a probabilistic graphical model to encapsulate these relationships.

Given these substantial similarities, we posit that there exists a potential for a unified framework that bridges the consciousness prior and our paradigm. Consequently, further exploration into establishing connections between these two paradigms and potentially crafting a more powerful framework is a topic ripe for investigation.

Bibliography

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[2] Minkyung Baek, Frank DiMaio, Ivan Anishchenko, Justas Dauparas, Sergey Ovchinnikov, Gyu Rie Lee, Jue Wang, Qian Cong, Lisa N Kinch, R Dustin Schaeffer, et al. Accurate prediction of protein structures and interactions using a three-track neural network. *Science*, 373(6557):871–876, 2021.

[3] Gökhan BakIr, Thomas Hofmann, Bernhard Schölkopf, Alexander J Smola, and Ben Taskar. *Predicting structured data*. MIT press, 2007.

[4] Ivana Balazevic, Carl Allen, and Timothy Hospedales. Tucker: Tensor factorization for knowledge graph completion. In *EMNLP*, 2019.

[5] Yoshua Bengio. The consciousness prior. *arXiv preprint arXiv:1709.08568*, 2017.

[6] Julian Besag. Statistical analysis of non-lattice data. *The statistician*, pages 179–195, 1975.

[7] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, 2008.

[8] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In *NeurIPS*, 2013.

[9] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[10] Robin Burke. Knowledge-based recommender systems. *Encyclopedia of library and information systems*, 2000.

[11] Liwei Cai and William Yang Wang. Kbgan: Adversarial learning for knowledge graph embeddings. In *NAACL*, 2018.

[12] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. Simple and deep graph convolutional networks. In *ICML*, 2020.

[13] Wenhu Chen, Wenhan Xiong, Xifeng Yan, and William Wang. Variational knowledge graph reasoning. In *NAACL*, 2018.

[14] Xinshi Chen, Yufei Zhang, Christoph Reisinger, and Le Song. Understanding deep architecture with reasoning layer. *NeurIPS*, 2020.

[15] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *KDD*, 2019.

[16] Kenneth Ward Church. A stochastic parts program and noun phrase parser for unrestricted text. In *ANLC*, 1988.

[17] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). In *ICLR*, 2016.

[18] William W Cohen, Fan Yang, and Kathryn Rivard Mazaitis. Tensorlog: Deep learning meets probabilistic databases. *Journal of Artificial Intelligence Research*, 2018.

[19] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. In *NeurIPS*, 2020.

[20] Vítor Santos Costa, David Page, Maleeha Qazi, and James Cussens. Clp (bn): Constraint logic programming for probabilistic knowledge. In *UAI*, 2002.

[21] James Cussens. Stochastic logic programs: sampling, inference and applications. In *UAI*, 2000.

[22] Hanjun Dai, Bo Dai, and Le Song. Discriminative embeddings of latent variable models for structured data. In *ICML*, 2016.

[23] Rajarshi Das, Shehzaad Dhuliawala, Manzil Zaheer, Luke Vilnis, Ishan Durugkar, Akshay Krishnamurthy, Alex Smola, and Andrew McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In *ICLR*, 2018.

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

[25] Yuntian Deng, Yoon Kim, Justin Chiu, Demi Guo, and Alexander Rush. Latent alignment and variational attention. In *NeurIPS*, 2018.

[26] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *AAAI*, 2018.

[27] Boyang Ding, Quan Wang, Bin Wang, and Li Guo. Improving knowledge graph embedding using simple constraints. In *ACL*, 2018.

[28] Jonathan Eckstein, Noam Goldberg, and Ai Kagawa. Rule-enhanced penalized regression by column generation using rectangular maximum agreement. In *ICML*, 2017.

[29] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[30] Thomas Finley and Thorsten Joachims. Training structural svms when exact inference is intractable. In *ICML*, 2008.

[31] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, 1999.

[32] Luis Antonio Galárraga, Christina Teflioudi, Katja Hose, and Fabian Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *WWW*, 2013.

[33] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *Proceedings of the 36th International Conference on Machine Learning*, 2019.

[34] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *KDD*, 2018.

[35] Victor Garcia Satorras, Zeynep Akata, and Max Welling. Combining generative and discriminative models for hybrid inference. In *NeurIPS*, 2019.

[36] Samuel Gershman and Noah Goodman. Amortized inference in probabilistic reasoning. In *CogSci*, 2014.

[37] Lise Getoor and Benjamin Taskar. *Introduction to Statistical Relational Learning*. MIT Press, 1 edition, 2007.

[38] Lise Getoor, Nir Friedman, Daphne Koller, and Benjamin Taskar. Learning probabilistic models of relational structure. In *ICML*, 2001.

[39] Lise Getoor, Eran Segal, Ben Taskar, and Daphne Koller. Probabilistic models of text and link structure for hypertext classification. In *IJCAI Workshop*, 2001.

[40] Joseph C Giarratano and Gary Riley. *Expert systems*. PWS publishing co., 1998.

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

[42] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *ICML*, 2017.

[43] Noam Goldberg and Jonathan Eckstein. Boosting classifiers with tightened l0-relaxation penalties. In *ICML*, 2010.

[44] Google. Freebase data dumps. `https://developers.google.com/freebase/data`, 2018.

[45] Shu Guo, Quan Wang, Lihong Wang, Bin Wang, and Li Guo. Knowledge graph embedding with iterative guidance from soft rules. In *AAAI*, 2018.

[46] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

[47] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017.

[48] Xuming He, Richard S Zemel, and Miguel Á Carreira-Perpiñán. Multiscale conditional random fields for image labeling. In *CVPR*, 2004.

[49] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 1997.

[50] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 2013.

[51] Peter Jackson. *Introduction to expert systems*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[52] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

[53] Rudolf Kadlec, Ondrej Bajgar, and Jan Kleindienst. Knowledge base completion: Baselines strike back. In *Workshop on Representation Learning for NLP*, 2017.

[54] Daniel Kahneman. *Thinking, fast and slow*. Farrar, Straus and Giroux, New York, NY, 2011.

[55] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.

[56] Kristian Kersting and Luc De Raedt. Towards combining inductive logic programming with bayesian networks. In *ICILP*, 2001.

[57] Tushar Khot, Sriraam Natarajan, Kristian Kersting, and Jude Shavlik. Learning markov logic networks via functional gradient boosting. In *ICDM*, 2011.

[58] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

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

[60] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

[61] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

[62] Stanley Kok and Pedro Domingos. Learning the structure of markov logic networks. In *ICML*, 2005.

[63] Stanley Kok and Pedro Domingos. Statistical predicate invention. In *ICML*, 2007.

[64] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[65] Daphne Koller and Avi Pfeffer. Probabilistic frame-based systems. In *AAAI/IAAI*, 1998.

[66] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *ICDM*, 2016.

[67] Srijan Kumar, Bryan Hooi, Disha Makhija, Mohit Kumar, Christos Faloutsos, and VS Subrahmanian. Rev2: Fraudulent user prediction in rating platforms. In *WSDM*, 2018.

[68] Timothee Lacroix, Nicolas Usunier, and Guillaume Obozinski. Canonical tensor decomposition for knowledge base completion. In *ICML*, 2018.

[69] John Lafferty, Andrew McCallum, and Fernando CN Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[70] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. In *NAACL-HLT*, 2016.

[71] Ni Lao and William W Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 2010.

[72] Ni Lao, Tom Mitchell, and William W Cohen. Random walk inference and learning in a large scale knowledge base. In *EMNLP*, 2011.

[73] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553): 436–444, 2015.

[74] Guohao Li, Matthias Muller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? In *ICCV*, 2019.

[75] Bill Yuchen Lin, Xinyue Chen, Jamin Chen, and Xiang Ren. Kagnet: Knowledge-aware graph networks for commonsense reasoning. In *EMNLP*, 2019.

[76] Xi Victoria Lin, Richard Socher, and Caiming Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *EMNLP*, 2018.

[77] Liyuan Liu, Jingbo Shang, Xiang Ren, Frank Xu, Huan Gui, Jian Peng, and Jiawei Han. Empower sequence labeling with task-aware neural language model. In *AAAI*, 2018.

[78] Jiaqi Ma, Weijing Tang, Ji Zhu, and Qiaozhu Mei. A flexible generative framework for graph-based semi-supervised learning. In *NeurIPS*, 2019.

[79] Jiaqi Ma, Bo Chang, Xuefei Zhang, and Qiaozhu Mei. Copulagnn: Towards integrating representational and correlational roles of graphs in graph neural networks. In *ICLR*, 2021.

[80] Tengfei Ma, Cao Xiao, Junyuan Shang, and Jimeng Sun. Cgnf: Conditional graph neural fields. *ICLR Submission*, 2018.

[81] Xuezhe Ma and Eduard Hovy. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *ACL*, 2016.

[82] Christian Meilicke, Melisachew Wudage Chekol, Daniel Ruffinelli, and Heiner Stuckenschmidt. Anytime bottom-up rule learning for knowledge graph completion. In *IJCAI*, 2019.

[83] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 1995.

[84] Pasquale Minervini, Sebastian Riedel, Pontus Stenetorp, Edward Grefenstette, and Tim Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In *ICML*, 2020.

[85] Kevin P Murphy, Yair Weiss, and Michael I Jordan. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, 1999.

[86] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[87] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML*, 2010.

[88] Sriraam Natarajan, Tushar Khot, Kristian Kersting, Bernd Gutmann, and Jude Shavlik. Boosting relational dependency networks. In *ICILP*, 2010.

[89] Radford M Neal and Geoffrey E Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998.

[90] Jennifer Neville and David Jensen. Relational dependency networks. *Journal of Machine Learning Research*, 2007.

[91] Maximilian Nickel, Kevin Murphy, Volker Tresp, and Evgeniy Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 2016.

[92] Maximilian Nickel, Lorenzo Rosasco, and Tomaso Poggio. Holographic embeddings of knowledge graphs. In *AAAI*, 2016.

[93] OpenAI. OpenAI: Introducing ChatGPT, 2022. URL `https://openai.com/blog/chatgpt`.

[94] OpenAI. OpenAI: GPT-4, 2023. URL `https://openai.com/research/gpt-4`.

[95] Manfred Opper and David Saad. *Advanced mean field methods: Theory and practice*. MIT press, 2001.

[96] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

[97] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *KDD*, 2014.

[98] David Poole. Probabilistic horn abduction and bayesian networks. *Artificial intelligence*, 1993.

[99] Alexandrin Popescul and Lyle H Ungar. Structural logistic regression for link analysis. *Departmental Papers (CIS)*, page 133, 2003.

[100] Siyuan Qi, Wenguan Wang, Baoxiong Jia, Jianbing Shen, and Song-Chun Zhu. Learning human-object interactions by graph parsing neural networks. In *ECCV*, 2018.

[101] Meng Qu and Jian Tang. Probabilistic logic neural networks for reasoning. In *NeurIPS*, 2019.

[102] Meng Qu, Xiang Ren, Yu Zhang, and Jiawei Han. Weakly-supervised relation extraction by pattern-enhanced embedding learning. In *WWW*, 2018.

[103] Meng Qu, Yoshua Bengio, and Jian Tang. Gmnn: Graph markov neural networks. In *ICML*, 2019.

[104] Meng Qu, Tianyu Gao, Louis-Pascal Xhonneux, and Jian Tang. Few-shot relation extraction via bayesian meta-learning on relation graphs. In *ICML*, 2020.

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

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

[107] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *NeurIPS*, 2017.

[108] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. In *NeurIPS*, 2019.

[109] Ruslan Salakhutdinov and Hugo Larochelle. Efficient learning of deep boltzmann machines. In *AISTATS*, 2010.

[110] Erik F Sang and Fien De Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*, 2003.

[111] Sunita Sarawagi and Rahul Gupta. Accurate max-margin training for structured output spaces. In *ICML*, 2008.

[112] Victor Garcia Satorras and Max Welling. Neural enhanced belief propagation on factor graphs. In *AISTATS*, 2021.

[113] Vıctor Garcia Satorras, Emiel Hoogeboom, and Max Welling. E (n) equivariant graph neural networks. In *ICML*, 2021.

[114] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In

European Semantic Web Conference, 2018.

[115] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[116] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *HLT-NAACL*, 2003.

[117] Yelong Shen, Jianshu Chen, Po-Sen Huang, Yuqing Guo, and Jianfeng Gao. M-walk: Learning to walk over graphs using monte carlo tree search. In *NeurIPS*, 2018.

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

[119] Parag Singla and Pedro Domingos. Discriminative training of markov logic networks. In *AAAI*, 2005.

[120] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[121] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *ICLR*, 2019.

[122] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *ACL*, 2020.

[123] Charles Sutton and Andrew McCallum. An introduction to conditional random fields for relational learning. *Introduction to statistical relational learning*, 2006.

[124] Charles Sutton and Andrew McCallum. Piecewise training for structured prediction. *Machine learning*, 2009.

[125] Charles Sutton and Andrew McCallum. *An Introduction to Conditional Random Fields*. Now Publishers Inc, 2012.

[126] Jie Tang, Jing Zhang, Limin Yao, Juanzi Li, Li Zhang, and Zhong Su. Arnetminer: extraction and mining of academic social networks. In *KDD*, 2008.

[127] Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *UAI*, 2002.

[128] Ben Taskar, Ming-Fai Wong, Pieter Abbeel, and Daphne Koller. Link prediction in relational data. In *NIPS*, 2004.

[129] Ben Taskar, Pieter Abbeel, Ming-Fai Wong, and Daphne Koller. Relational markov networks. *Introduction to statistical relational learning*, 2007.

[130] Komal K Teru and William L Hamilton. Inductive relation prediction on knowledge graphs. In *ICML*, 2020.

[131] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine*

learning, 4(2):26–31, 2012.

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

[133] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[134] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[135] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *ICML*, 2016.

[136] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 2005.

[137] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018.

[138] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. In *ICLR*, 2019.

[139] Martin J Wainwright and Michael Irwin Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[140] Martin J Wainwright, Tommi S Jaakkola, and Alan S Willsky. Tree-reweighted belief propagation algorithms and approximate ml estimation by pseudo-moment matching. In *AISTATS*, 2003.

[141] Binghui Wang, Jinyuan Jia, and Neil Zhenqiang Gong. Semi-supervised node classification on graphs: Markov random fields vs. graph neural networks. In *AAAI*, 2021.

[142] Hongwei Wang, Fuzheng Zhang, Jialin Wang, Miao Zhao, Wenjie Li, Xing Xie, and Minyi Guo. Ripplenet: Propagating user preferences on the knowledge graph for recommender systems. In *CIKM*, 2018.

[143] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *CIKM*, 2013.

[144] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Proppr: Efficient first-order probabilistic logic programming for structure discovery, parameter learning,

and scalable inference. In *Workshops at AAAI*, 2014.

[145] William Yang Wang, Kathryn Mazaitis, and William W Cohen. Structure learning via parameter learning. In *CIKM*, 2014.

[146] Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *AAAI*, 2014.

[147] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*, 2022.

[148] Yair Weiss and William T Freeman. On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 2001.

[149] Michael P Wellman, John S Breese, and Robert P Goldman. From knowledge bases to decision models. *The Knowledge Engineering Review*, 1992.

[150] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 1992.

[151] Louis-Pascal Xhonneux, Meng Qu, and Jian Tang. Continuous graph neural networks. In *ICML*, 2020.

[152] Rongjing Xiang and Jennifer Neville. Pseudolikelihood em for within-network relational learning. In *ICDM*, 2008.

[153] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *EMNLP*, 2017.

[154] Minghao Xu, Meng Qu, Bingbing Ni, and Jian Tang. Joint modeling of visual objects and relations for scene graph generation. In *NeurIPS*, 2021.

[155] Rong Xu and QuanQiu Wang. Large-scale extraction of accurate drug-disease treatment pairs from biomedical literature for drug repurposing. *BMC bioinformatics*, 2013.

[156] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. *ICLR*, 2015.

[157] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *NeurIPS*, 2017.

[158] Yuan Yang and Le Song. Learn to explain efficiently via neural logic inductive learning. In *ICLR*, 2020.

[159] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *ICML*, 2016.

[160] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

[161] Xuchen Yao and Benjamin Van Durme. Information extraction over structured data: Question answering with freebase. In *ACL*, 2014.

[162] Jonathan S Yedidia, William T Freeman, and Yair Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on information theory*, 2005.

[163] KiJung Yoon, Renjie Liao, Yuwen Xiong, Lisa Zhang, Ethan Fetaya, Raquel Urtasun, Richard Zemel, and Xaq Pitkow. Inference in probabilistic graphical models by graph neural networks. In *ICLR Workshop*, 2018.

[164] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graphsaint: Graph sampling based inductive learning method. In *ICLR*, 2020.

[165] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Can graph neural networks help logic reasoning? *arXiv preprint arXiv:1906.02111*, 2019.

[166] Yuyu Zhang, Xinshi Chen, Yuan Yang, Arun Ramamurthy, Bo Li, Yuan Qi, and Le Song. Efficient probabilistic logic reasoning with graph neural networks. In *ICLR*, 2020.

[167] Jianan Zhao, Meng Qu, Chaozhuo Li, Hao Yan, Qian Liu, Rui Li, Xing Xie, and Jian Tang. Learning on large-scale text-attributed graphs via variational inference. In *ICLR*, 2023.

[168] Zilong Zheng, Wenguan Wang, Siyuan Qi, and Song-Chun Zhu. Reasoning visual dialogs with structural and partial observations. In *CVPR*, 2019.

[169] Dengyong Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *NIPS*, 2004.

[170] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *ICML*, 2003.

[171] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In *NeurIPS*, 2021.

[172] Marinka Zitnik and Jure Leskovec. Predicting multicellular function through multi-layer tissue networks. In *ISMB*, 2017.

Chapter A

GMNN: Graph Markov Neural Networks

A.1. Optimality Condition of the Inference Network

Theorem A.1.1. *For a node n and a fixed variational distribution $q_\theta(\mathbf{y}_{NB(n)\cap U}|\mathbf{x}_V)$ for nodes in $NB(n) \cap U$, the optimum of $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$, denoted by $q^*(\mathbf{y}_n|\mathbf{x}_V)$, is characterized by the following condition:*

$$\log q^*(\mathbf{y}_n|\mathbf{x}_V) = \mathbb{E}_{q_\theta(\mathbf{y}_{NB(n)\cap U}|\mathbf{x}_V)}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{NB(n)},\mathbf{x}_V)] + const.$$

Proof. To make the notation concise, we omit $\mathbf{x}_V$ in the proof (e.g. simplifying $q_\theta(\mathbf{y}_n|\mathbf{x}_V)$ as $q_\theta(\mathbf{y}_n)$). Our goal for $q_\theta(\mathbf{y}_n)$ is to minimize the KL divergence between $q_\theta(\mathbf{y}_U)$ and $p_\phi(\mathbf{y}_U|\mathbf{y}_L)$. Therefore, the objective for $q_\theta(\mathbf{y}_n)$ is formulated as follows:

$$\begin{aligned}
O(q_\theta(\mathbf{y}_n)) &= -\operatorname{KL}(q_\theta(\mathbf{y}_U)||p_\phi(\mathbf{y}_U|\mathbf{y}_L)) \\
&= \sum_{\mathbf{y}_U} q_\theta(\mathbf{y}_U)[\log p_\phi(\mathbf{y}_U|\mathbf{y}_L) - \log q_\theta(\mathbf{y}_U)] \\
&= \sum_{\mathbf{y}_U} \left(\prod_{n'} q_\theta(\mathbf{y}_{n'})\right) \left[\log p_\phi(\mathbf{y}_U,\mathbf{y}_L) - \sum_{n'} \log q_\theta(\mathbf{y}_{n'})\right] + \text{const} \\
&= \sum_{\mathbf{y}_n} \sum_{\mathbf{y}_{U\setminus n}} \left(q_\theta(\mathbf{y}_n) \prod_{n'\neq n} q_\theta(\mathbf{y}_{n'})\right) \left[\log p_\phi(\mathbf{y}_U,\mathbf{y}_L) - \sum_{n'} \log q_\theta(\mathbf{y}_{n'})\right] + \text{const} \\
&= \sum_{\mathbf{y}_n} q_\theta(\mathbf{y}_n) \sum_{\mathbf{y}_{U\setminus n}} \prod_{n'\neq n} q_\theta(\mathbf{y}_{n'}) \log p_\phi(\mathbf{y}_U,\mathbf{y}_L) - \\
&\quad \sum_{\mathbf{y}_n} q_\theta(\mathbf{y}_n) \sum_{\mathbf{y}_{U\setminus n}} \prod_{n'\neq n} q_\theta(\mathbf{y}_{n'}) \left[\sum_{n'\neq n} \log q_\theta(\mathbf{y}_{n'}) + \log q_\theta(\mathbf{y}_n)\right] + \text{const} \\
&= \sum_{\mathbf{y}_n} q_\theta(\mathbf{y}_n) \log \mathcal{F}(\mathbf{y}_n) - \sum_{\mathbf{y}_n} q_\theta(\mathbf{y}_n) \log q_\theta(\mathbf{y}_n) + \text{const} \\
&= -\operatorname{KL}\left(q_\theta(\mathbf{y}_n)||\frac{\mathcal{F}(\mathbf{y}_n)}{Z}\right) + \text{const.}
\end{aligned}$$

Here, Z is a normalization term making $\mathcal{F}(\mathbf{y}_n)$ a valid distribution on $\mathbf{y}_n$, and we have:

$$\log \mathcal{F}(\mathbf{y}_n) = \sum_{\mathbf{y}_{U\setminus n}} \prod_{n' \neq n} q_\theta(\mathbf{y}_{n'}) \log p_\phi(\mathbf{y}_U, \mathbf{y}_L) = \mathbb{E}_{q_\theta(\mathbf{y}_{U\setminus n})}[\log p_\phi(\mathbf{y}_U, \mathbf{y}_L)].$$

Based on the above mathematical manipulation of the objective function $O(q_\theta(\mathbf{y}_n))$, if a local optimal of $q_\theta(\mathbf{y}_n)$ is denoted by $q^*(\mathbf{y}_n|\mathbf{x}_V)$, then $q^*(\mathbf{y}_n|\mathbf{x}_V)$ must be equal to $\frac{\mathcal{F}(\mathbf{y}_n)}{Z}$, and thus we have:

$$\begin{aligned}
\log q^*(\mathbf{y}_n) &= \log \mathcal{F}(\mathbf{y}_n) + \text{const} \\
&= \mathbb{E}_{q_\theta(\mathbf{y}_{U\setminus n})}[\log p_\phi(\mathbf{y}_U, \mathbf{y}_L)] + \text{const} \\
&= \mathbb{E}_{q_\theta(\mathbf{y}_{U\setminus n})}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{V\setminus n})] + \text{const} \\
&= \mathbb{E}_{q_\theta(\mathbf{y}_{U\setminus n})}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)})] + \text{const} \\
&= \mathbb{E}_{q_\theta(\mathbf{y}_{\mathrm{NB}(n)\cap U})}[\log p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)})] + \text{const}.
\end{aligned}$$

Here, $p_\phi(\mathbf{y}_n|\mathbf{y}_{V\setminus n}) = p_\phi(\mathbf{y}_n|\mathbf{y}_{\mathrm{NB}(n)})$ is based on the conditional independence property of Markov networks. □

Chapter B

SPN: Structured Proxy Networks

B.1. Derivation of the Maximin Game

As discussed in the Section 4.3, optimizing the joint label distribution p_θ to maximize the log-likelihood $\log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E)$ on a training graph $(\mathbf{y}_V^*,\mathbf{x}_V, E)$ is equivalent to solving a maximin game. In this section, we provide the detailed derivation.

Let $\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)$ be the potential function as below:

$$\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) = \exp\left\{\sum_{s\in V} \theta_s(y_s,\mathbf{x}_V, E) + \sum_{(s,t)\in E} \theta_{st}(y_s, y_t,\mathbf{x}_V, E)\right\}. \tag{B.1.1}$$

For each training graph $(\mathbf{y}_V^*,\mathbf{x}_V, E)$, we aim at maximizing the following log-likelihood function:

$$\begin{aligned}\log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) &= \log \frac{1}{Z_\theta(\mathbf{x}_V, E)}\psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) \\ &= \log \psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) - \log Z_\theta(\mathbf{x}_V, E) \\ &= \log \psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) - \log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E).\end{aligned} \tag{B.1.2}$$

However, the term $\log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)$ is computationally intractable, as we need to sum over all the possible $\mathbf{y}_V$. To solve the problem, we introduce a variational joint distribution $q(\mathbf{y}_V)$ defined on all node labels $\mathbf{y}_V$, and use the Jensen's inequality to derive an estimation of the term $\log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)$ as follows:

$$\begin{aligned}\log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) &= \log \mathbb{E}_{q(\mathbf{y}_V)}\left[\frac{\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{q(\mathbf{y}_V)}\right] \\ &\geq \mathbb{E}_{q(\mathbf{y}_V)}\left[\log \frac{\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{q(\mathbf{y}_V)}\right] \\ &= \mathbb{E}_{q(\mathbf{y}_V)}[\log \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)] - \mathbb{E}_{q(\mathbf{y}_V)}[\log q(\mathbf{y}_V)].\end{aligned} \tag{B.1.3}$$

The equation holds if and only if $q(\mathbf{y}_V) = p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)$, and hence:

$$\log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) = \max_{q(\mathbf{y}_V)} \left\{ \mathbb{E}_{q(\mathbf{y}_V)}[\log \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)] - \mathbb{E}_{q(\mathbf{y}_V)}[\log q(\mathbf{y}_V)] \right\}. \tag{B.1.4}$$

By taking the above result into Equation (B.1.2), we obtain:

$$\begin{aligned} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) &= \log \psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) - \log \sum_{\mathbf{y}_V} \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) \\ &= \min_{q(\mathbf{y}_V)} \left\{ \log \psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) - \mathbb{E}_{q(\mathbf{y}_V)}[\log \psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)] + \mathbb{E}_{q(\mathbf{y}_V)}[\log q(\mathbf{y}_V)] \right\}. \end{aligned} \tag{B.1.5}$$

As $\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) = \exp\{\sum_{s\in V} \theta_s(y_s, \mathbf{x}_V, E) + \sum_{(s,t)\in E} \theta_{st}(y_s, y_t, \mathbf{x}_V, E)\}$, we have:

$$\log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) = \min_q \mathcal{L}(\theta, q), \tag{B.1.6}$$

with:

$$\begin{aligned} \mathcal{L}(\theta, q) &= -H[q(\mathbf{y}_V)] \\ &\quad + \sum_{(s,t)\in E} \{\theta_{st}(y_s^*, y_t^*) - \mathbb{E}_{q_{st}(y_s,y_t)}[\theta_{st}(y_s, y_t)]\} + \sum_{s\in V} \{\theta_s(y_s^*) - \mathbb{E}_{q_s(y_s)}[\theta_s(y_s)]\}. \end{aligned} \tag{B.1.7}$$

Therefore, optimizing θ to maximize the log-likelihood function is equivalent to solving the following maximin game:

$$\max_\theta \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) = \max_\theta \min_q \mathcal{L}(\theta, q). \tag{B.1.8}$$

B.2. Derivation of the Moment-matching Conditions

In the CRF model defined in the preliminary section, the parameter θ consists of the output values of all θ-functions. In other words, $\theta = \{\theta_s(y_s)\}_{y_s\in\mathcal{Y}, s\in V} \cup \{\theta_{st}(y_s, y_t)\}_{y_s\in\mathcal{Y}, y_t\in\mathcal{Y}, s\in V}$, where $\mathcal{Y}$ is the set of all the possible node labels.

By definition, $p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)$ belongs to the exponential family. According to properties of exponential family distributions, $\log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E)$ is strictly concave with respect to θ. Therefore, the optimal θ is unique, which is characterized by the condition of $\frac{\partial}{\partial\theta} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) = 0$. Formally, $\frac{\partial}{\partial\theta} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E)$ can be computed as below:

$$\frac{\partial}{\partial\theta_s(\hat{y}_s)} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) = \frac{\partial}{\partial\theta} \log \psi_\theta(\mathbf{y}_V^*, \mathbf{x}_V, E) - \frac{\partial}{\partial\theta} \log Z_\theta(\mathbf{x}_V, E). \tag{B.2.1}$$

For $\frac{\partial}{\partial\theta}\log Z_\theta(\mathbf{x}_V, E)$, we have:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log Z_\theta(\mathbf{x}_V, E) &= \frac{\partial}{\partial\theta}\log\sum_{\mathbf{y}_V}\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) \\
&= \frac{\sum_{\mathbf{y}_V}\frac{\partial}{\partial\theta}\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{\sum_{\mathbf{y}_V}\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)} \\
&= \frac{\sum_{\mathbf{y}_V}\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{\sum_{\mathbf{y}_V}\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)} \\
&= \sum_{\mathbf{y}_V}\left[\frac{\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{\sum_{\mathbf{y}'_V}\psi_\theta(\mathbf{y}'_V, \mathbf{x}_V, E)}\frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\right] \\
&= \sum_{\mathbf{y}_V}\left[\frac{\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)}{Z_\theta}\frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\right] \\
&= \mathbb{E}_{p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)}\left[\frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\right].
\end{aligned} \tag{B.2.2}$$

By combining the above two equations, we have:

$$\frac{\partial}{\partial\theta}\log p_\theta(\mathbf{y}^*_V|\mathbf{x}_V, E) = \frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}^*_V, \mathbf{x}_V, E) - \mathbb{E}_{p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)}\left[\frac{\partial}{\partial\theta}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\right]. \tag{B.2.3}$$

The potential function ψ_θ above is defined as $\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E) = \exp\{\sum_{s\in V}\theta_s(y_s,\mathbf{x}_V, E) + \sum_{(s,t)\in E}\theta_{st}(y_s, y_t,\mathbf{x}_V, E)\}$. If we consider each specific scalar $\theta_s(\hat{y}_s)$, and taking the derivative with respect to the scalar to 0, we obtain:

$$\begin{aligned}
0 &= \frac{\partial}{\partial\theta_s(\hat{y}_s)}\log p_\theta(\mathbf{y}^*_V|\mathbf{x}_V, E) \\
&= \frac{\partial}{\partial\theta_s(\hat{y}_s)}\log\psi_\theta(\mathbf{y}^*_V, \mathbf{x}_V, E) - \mathbb{E}_{p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)}\left[\frac{\partial}{\partial\theta_s(\hat{y}_s)}\log\psi_\theta(\mathbf{y}_V, \mathbf{x}_V, E)\right] \\
&= \mathbb{I}_{y^*_s}\{\hat{y}_s\}\left[\frac{\partial}{\partial\theta_s(\hat{y}_s)}\theta_s(\hat{y}_s)\right] - \sum_{\mathbf{y}_V}p_\theta(\mathbf{y}_V|\mathbf{x}_V, E)\left[\mathbb{I}_{y^*_s}\{\hat{y}_s\}\frac{\partial}{\partial\theta_s(\hat{y}_s)}\theta_s(\hat{y}_s)\right] \\
&= \mathbb{I}_{y^*_s}\{\hat{y}_s\}\left[\frac{\partial}{\partial\theta_s(\hat{y}_s)}\theta_s(\hat{y}_s)\right] - p_\theta(\hat{y}_s|\mathbf{x}_V, E)\left[\frac{\partial}{\partial\theta_s(\hat{y}_s)}\theta_s(\hat{y}_s)\right] \\
&= \mathbb{I}_{y^*_s}\{\hat{y}_s\} - p_\theta(\hat{y}_s|\mathbf{x}_V, E),
\end{aligned} \tag{B.2.4}$$

which implies $p_\theta(\hat{y}_s|\mathbf{x}_V, E) = \mathbb{I}_{y^*_s}\{\hat{y}_s\}$ for the optimal θ. Moreover, this equation holds for all $s \in V$ and all $\hat{y}_s \in \mathcal{Y}$.

Similarly, for each scalar $\theta_{st}(\hat{y}_s, \hat{y}_t)$, we have that $\frac{\partial}{\partial\theta_{st}(\hat{y}_s,\hat{y}_t)}\log p_\theta(\mathbf{y}^*_V|\mathbf{x}_V, E) = 0$ is equivalent to $p_\theta(\hat{y}_s,\hat{y}_t|\mathbf{x}_V, E) = \mathbb{I}_{y^*_s,y^*_t}\{\hat{y}_s, \hat{y}_t\}$. This equation holds for all $(s,t) \in E$ and all the choices of $(\hat{y}_s, \hat{y}_t) \in \mathcal{Y}\times\mathcal{Y}$.

Therefore, the optimal θ-functions are characterized by the moment-matching conditions as below:

$$p_\theta(y_s|\mathbf{x}_V, E) = \mathbb{I}_{y_s^*}\{y_s\} \quad \forall s \in V, \quad p_\theta(y_s,y_t|\mathbf{x}_V, E) = \mathbb{I}_{y_s^*,y_t^*}\{y_s, y_t\} \quad \forall (s,t) \in E. \tag{B.2.5}$$

B.3. Proof of Proposition 1 in SPN

Next, we prove Proposition 1. We first restate the proposition as follows:

Proposition *Consider a set of nonzero pseudomarginals* $\{\tau_s(y_s)\}_{s\in V}$ *and* $\{\tau_{st}(y_s, y_t)\}_{(st)\in E}$ *which satisfy* $\sum_{y_s} \tau_{st}(y_s, y_t) = \tau_t(y_t)$ *and* $\sum_{y_t} \tau_{st}(y_s, y_t) = \tau_s(y_s)$ *for all* $(s,t) \in E$.
If we parameterize the θ*-functions of* p_θ *in Equation* (4.3.2) *in the following way:*

$$\theta_s(y_s) = \log \tau_s(y_s) \quad \forall s \in V, \quad \theta_{st}(y_s,y_t) = \log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)\tau_t(y_t)} \quad \forall (s,t) \in E, \tag{B.3.1}$$

then $\{\tau_s(y_s)\}_{s\in V}$ *and* $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$ *are specified by a fixed point of the sum-product loopy belief propagation algorithm when applied to the joint distribution* p_θ*, which implies that:*

$$\tau_s(y_s) \approx p_\theta(y_s) \quad \forall s \in V, \quad \tau_{st}(y_s, y_t) \approx p_\theta(y_s,y_t) \quad \forall (s,t) \in E. \tag{B.3.2}$$

Proof: To prove the proposition, we first summarize the workflow of the sum-product loopy belief propagation algorithm. In sum-product loopy belief propagation, we introduce a message function $m_{t\to s}(y_s)$ for each edge $(s,t) \in E$. Then we iteratively update all message functions as follows:

$$m_{t\to s}(y_s) \propto \sum_{y_t} \left\{ \exp(\theta_t(y_t) + \theta_{st}(y_s, y_t)) \prod_{s'\in N(t)\setminus s} m_{s'\to t}(y_t) \right\}, \tag{B.3.3}$$

where $N(t)$ represents the set of neighbors for node t.

Once the process converges or after sufficient iterations, the approximation of the node marginals and the edge marginals (i.e., $\{q_s(y_s)\}_{s\in V}$ and $\{q_{st}(y_s, y_t)\}_{(s,t)\in E}$) can be recovered by the message functions $\{m_{t\to s}(y_s)\}_{(s,t)\in E}$ as follows:

$$q_s(y_s) \propto \exp(\theta_s(y_s)) \prod_{t\in N(s)} m_{t\to s}(y_s), \tag{B.3.4}$$

$$q_{st}(y_s, y_t) \propto \exp(\theta_s(y_s) + \theta_t(y_t) + \theta_{st}(y_s, y_t)) \prod_{t'\in N(s)\setminus t} m_{t'\to s}(y_s) \prod_{s'\in N(t)\setminus s} m_{s'\to t}(y_t). \tag{B.3.5}$$

Next, let us move back to our case, where we parameterize the θ-functions with a set of pseudomarginals as in Equation (B.3.1). For such a specific parameterization of the θ-functions, we claim that one fixed point of Equation (B.3.3) is achieved when $m_{t\to s}(y_s) = 1$ for all $(s,t) \in E$. To prove that, we notice that when all the message functions equal to 1,

the left side of Equation (B.3.3) is apparently 1. The right side of Equation (B.3.3) can be computed as below:

$$\begin{aligned}
&\sum_{y_t} \exp(\theta_t(y_t) + \theta_{st}(y_s, y_t)) \prod_{s' \in N(t) \setminus s} m_{s' \to t}(y_t) \\
=&\sum_{y_t} \exp(\theta_t(y_t) + \theta_{st}(y_s, y_t)) \\
=&\sum_{y_t} \exp\left(\log \tau_t(y_t) + \log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)\tau_t(y_t)}\right) \\
=&\sum_{y_t} \exp\left(\log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)}\right) \\
=&\sum_{y_t} \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)} \\
=&\frac{\tau_s(y_s)}{\tau_s(y_s)} \\
=&1.
\end{aligned} \tag{B.3.6}$$

We can see that both the left side and the right side of Equation (B.3.3) are 1, and hence $\{m_{t \to s}(y_s) = 1\}_{(s,t) \in E}$ specifies a fixed point of sum-product loopy belief propagation. For this fixed point, $q_s(y_s)$ can be computed as follows:

$$q_s(y_s) \propto \exp(\theta_s(y_s)) \prod_{t \in N(s)} m_{t \to s}(y_s) = \exp(\theta_s(y_s)) = \tau_s(y_s). \tag{B.3.7}$$

Similarly, we can compute $q_{st}(y_s, y_t)$ as:

$$\begin{aligned}
q_{st}(y_s, y_t) &\propto \exp(\theta_s(y_s) + \theta_t(y_t) + \theta_{st}(y_s, y_t)) \prod_{t' \in N(s) \setminus t} m_{t' \to s}(y_s) \prod_{s' \in N(t) \setminus s} m_{s' \to t}(y_t) \\
&= \exp(\theta_s(y_s) + \theta_t(y_t) + \theta_{st}(y_s, y_t)) \\
&= \exp\left(\log \tau_s(y_s) + \log \tau_t(y_t) + \log \frac{\tau_{st}(y_s, y_t)}{\tau_s(y_s)\tau_t(y_t)}\right) \\
&= \tau_{st}(y_s, y_t).
\end{aligned} \tag{B.3.8}$$

From the above two equations, we can see that $\{\tau_s(y_s)\}_{s \in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t) \in E}$ are specified by a fixed point (i.e., $m_{t \to s}(y_s) = 1$ for all $(s, t) \in E$) of sum-product loopy belief propagation. As sum-product loopy belief propagation often works well in practice to approximate the marginal distributions on nodes and edges, we thus have $\tau_s(y_s) \approx p_\theta(y_s)$ for each node and $\tau_{st}(y_s, y_t) \approx p_\theta(y_s,y_t)$ for each edge.

B.4. Solving the Proxy Problem with Constrained Optimization

The key innovation of our proposed approach is on the proxy optimization problem which is used to approximate the original learning problem. Formally, the proxy optimization problem is stated as:

$$\begin{aligned} &\min_{\tau,\theta} \sum_{s\in V} d\left(\mathbb{I}_{y_s^*}\{y_s\}, \tau_s(y_s)\right) + \sum_{(s,t)\in E} d\left(\mathbb{I}_{(y_s^*,y_t^*)}\{(y_s,y_t)\}, \tau_{st}(y_s,y_t)\right), \\ &\text{subject to} \quad \theta_s = \log \tau_s(y_s), \quad \theta_{st}(y_s,y_t) = \log \frac{\tau_{st}(y_s,y_t)}{\tau_s(y_s)\tau_t(y_t)}, \\ &\quad \text{and} \quad \sum_{y_s} \tau_{st}(y_s,y_t) = \tau_t(y_t), \quad \sum_{y_t} \tau_{st}(y_s,y_t) = \tau_s(y_s), \end{aligned} \tag{B.4.1}$$

for all nodes and edges, where d can be any divergence measure between two distributions.

In our implementation, we ignore these consistency constraints, i.e., $\sum_{y_s} \tau_{st}(y_s,y_t) = \tau_t(y_t)$ and $\sum_{y_t} \tau_{st}(y_s,y_t) = \tau_s(y_s)$. This ie because by by optimizing the objective, the obtained pseudomarginals τ would well approximate the observed node and edge marginals, i.e., $\tau_s(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ and $\tau_{st}(y_s,y_t) \approx \mathbb{I}_{(y_s^*,y_t^*)}\{(y_s,y_t)\}$, and hence τ would almost naturally satisfy the constraints.

To demonstrate ignoring the consistency constraint makes sense, we also tried a constrained optimization method for solving the proxy problem. Specifically, we add a quadratic term to penalize the inconsistency between $\tau_{st}(y_s,y_t)$ and $\tau_s(y_s)$ as well as $\tau_t(y_t)$, resulting in the following problem:

$$\begin{aligned} &\min_{\tau,\theta} \sum_{s\in V} d\left(\mathbb{I}_{y_s^*}\{y_s\}, \tau_s(y_s)\right) + \sum_{(s,t)\in E} d\left(\mathbb{I}_{(y_s^*,y_t^*)}\{(y_s,y_t)\}, \tau_{st}(y_s,y_t)\right) \\ &\quad + \alpha \sum_{(s,t)\in E} \left[\sum_{y_t}\{\sum_{y_s} \tau_{st}(y_s,y_t) - \tau_t(y_t)\}^2 + \sum_{y_s}\{\sum_{y_t} \tau_{st}(y_s,y_t) - \tau_s(y_s)\}^2\right], \\ &\quad \text{subject to} \quad \theta_s = \log \tau_s(y_s), \quad \theta_{st}(y_s,y_t) = \log \frac{\tau_{st}(y_s,y_t)}{\tau_s(y_s)\tau_t(y_t)}, \end{aligned} \tag{B.4.2}$$

for all nodes and edges. Again, d is a divergence measure between two distributions, and we choose to use the KL divergence. α is a hyperparameter deciding the weight of the penalty term.

Algorithm	Constrained Optimization	Cora*	Citeseer*	Pubmed*
SPN-GAT	w/o	**49.10** ± 3.80	**42.89** ± 1.30	**47.79 ± 1.33**
	with	48.83 ± 3.51	42.04 ± 1.23	47.55 ± 1.24

Table 33. Analysis of constrained optimization methods in SPN.

We conduct empirical comparison of this constrained optimization method and our default implementation where the consistency constraint is ignored. The results are presented in Table 33. We can see that the constrained optimization method does not lead to improvement, which shows that ignoring the consistency constraint is empirically reasonable.

B.5. Understanding SPNs as Optimizing a Surrogate for the Log-likelihood

In the model section, we motivate SPNs from the moment-matching conditions of the optimal θ-functions. Specifically, we initialize the θ-functions at a state where the moment-matching conditions are approximately satisfied, yielding a near-optimal joint distribution. Then we further tune the θ-functions to solve the maximin game. Besides this perspective, SPNs can also be understood as optimizing a surrogate for the log-likelihood function. Next, we introduce the details.

Remember that maximizing the log-likelihood function is equivalent to solving a maximin game as:

$$\begin{aligned}\max_{\theta} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) = \max_{\theta}\min_{q} \mathcal{L}(\theta, q), \quad \text{with} \quad \mathcal{L}(\theta, q) = -H[q(\mathbf{y}_V)] \\ + \sum_{s\in V}\{\theta_s(y_s^*) - \mathbb{E}_{q_s(y_s)}[\theta_s(y_s)]\} + \sum_{(s,t)\in E}\{\theta_{st}(y_s^*, y_t^*) - \mathbb{E}_{q_{st}(y_s,y_t)}[\theta_{st}(y_s, y_t)]\}.\end{aligned} \tag{B.5.1}$$

Here, $q(\mathbf{y}_V)$ is a joint distribution on all the node labels. $q_s(y_s)$ and $q_{st}(y_s,y_t)$ are the corresponding marginal distributions.

Although the above maximin game is equivalent to the original problem of maximizing likelihood, solving the maximin game is nontrivial. In particular, there are two key challenges, i.e., (1) how to specify constraints to characterize a valid joint distribution $q(\mathbf{y}_V)$ and (2) how to compute its entropy $H(q) = -\mathbb{E}_{q(\mathbf{y}_V)}[\log q(\mathbf{y}_V)]$. To deal with the challenge, a common practice used in loopy belief propagation is to make the following two approximations:

(1) Instead of specifying constraints to let $q(\mathbf{y}_V)$ be a valid joint distribution, we introduce a set of pseudomarginals as approximation to a valid joint distribution. Specifically, these pseudomarginals are denoted as $\tilde{q} = \{q_s(y_s)\}_{s\in V} \cup \{q_{st}(y_s, y_t)\}_{(s,t)\in E}$, and they satisfy $\sum_{y_s} q_{st}(y_s, y_t) = q_t(y_t)$ and $\sum_{y_t} q_{st}(y_s, y_t) = q_s(y_s)$ for all $(s,t) \in E$.

(2) We approximate the entropy $H(q)$ with Bethe entropy approximation $H_{\text{Bethe}}(\tilde{q})$, which is defined as follows:

$$H_{\text{Bethe}}(\tilde{q}) = -\sum_{s\in V}\mathbb{E}_{q_s(y_s)}[\log q_s(y_s)] - \sum_{(s,t)\in E}\mathbb{E}_{q_{st}(y_s,y_t)}\left[\log \frac{q_{st}(y_s, y_t)}{q_s(y_s)q_t(y_t)}\right]. \tag{B.5.2}$$

With the two approximations, we get the following maximin game as a surrogate for the likelihood maximization problem:

$$\max_{\theta} \log p_\theta(\mathbf{y}_V^*|\mathbf{x}_V, E) \approx \max_{\theta} \min_{\tilde{q}} \mathcal{L}_{\text{Bethe}}(\theta, \tilde{q}), \tag{B.5.3}$$

with:

$$\begin{aligned}\mathcal{L}_{\text{Bethe}}(\theta,\tilde{q}) = &-H_{\text{Bethe}}(\tilde{q}) \\ &+ \sum_{(s,t)\in E} \{\theta_{s,t}(y_s^*, y_t^*) - \mathbb{E}_{q_{st}(y_s,y_t)}[\theta_{s,t}(y_s, y_t)]\} + \sum_{s\in V} \{\theta_s(y_s^*) - \mathbb{E}_{q_s(y_s)}[\theta_s(y_s)]\}.\end{aligned} \tag{B.5.4}$$

This problem is known as the Bethe variational problem (BVP) [139].

Such a problem can be solved by coordinate descent, where we alternate between updating $\tilde{q}$ to minimize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$ and updating θ to maximize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$. According to Yedidia et al. [162], updating $\tilde{q}$ to minimize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$ can be exactly achieved by running sum-product loopy belief propagation on p_θ, where a fixed point of the belief propagation algorithm yields a local optima of $\tilde{q}$. On the other hand, updating θ to maximize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$ can be easily achieved by gradient ascent.

In addition to that, a stationary point $(\theta^*, \tilde{q}^*)$ of the above BVP is specified by following conditions:

$$\frac{\partial \mathcal{L}_{\text{Bethe}}(\theta^*, \tilde{q}^*)}{\partial \tilde{q}^*} = 0 \qquad \frac{\partial \mathcal{L}_{\text{Bethe}}(\theta^*, \tilde{q}^*)}{\partial \theta^*} = 0. \tag{B.5.5}$$

According to Yedidia et al. [162] and Wainwright and Jordan [139], the first condition is equivalent to the condition that $\tilde{q}^*$ is specified by a fixed-point of sum-product loopy belief propagation. The second condition states that the moment-matching conditions are satisfied, i.e., $q_s(y_s) = \mathbb{I}_{y_s^*}\{y_s\}$ on each node and $q_{st}(y_s,y_t) = \mathbb{I}_{(y_s^*,y_t^*)}\{(y_s, y_t)\}$ on each edge.

For our proposed approach SPN, it can be viewed as solving the BVP as defined in Equation (B.5.3). Through solving the proxy problem, SPN initializes θ at a state where the conditions of stationary points in Equation (B.5.5) are approximately satisfied. Then the fine-tuning stage of SPN further adjusts θ to solve the maximin game by alternatively updating θ and $\tilde{q}$.

More specifically, when solving the proxy optimization problem, by initializing θ in the way defined by Equation (B.3.1), the collection of pseudomarginal distributions $\{\tau_s(y_s)\}_{s\in V}$ and $\{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$ is specified by a fixed point of sum-product loopy belief propagation according to Proposition 1. This implies that $\frac{\partial}{\partial \tilde{q}} \mathcal{L}_{\text{Bethe}}(\theta, \tilde{q}) = 0$ for $\tilde{q} = \{\tau_s(y_s)\}_{s\in V} \cup \{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$. Meanwhile, as $\{\tau_s\}_{s\in V}$ and $\{\tau_{st}\}_{(s,t)\in E}$ are learned to match the true labels $\mathbf{y}_V^*$ on each training graph, we thus have $\tau_s(y_s) \approx \mathbb{I}_{y_s^*}\{y_s\}$ on each node and $\tau_{st}(y_s,y_t) \approx \mathbb{I}_{(y_s^*,y_t^*)}\{(y_s, y_t)\}$ on each edge. Therefore, the conditions in Equation (B.5.5) are approximately satisfied by $(\theta, \tilde{q})$ with $\tilde{q} = \{\tau_s(y_s)\}_{s\in V} \cup \{\tau_{st}(y_s, y_t)\}_{(s,t)\in E}$, which means that solving the proxy problem yields a θ to roughly match the conditions of stationary

points for the BVP in Equation (B.5.3). Afterwards, the refinement stage of SPN is exactly trying to solve the maximin game of BVP in Equation (B.5.3), where we alternate between updating $\tilde{q}$ to minimize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$ via sum-product loopy belief propagation and updating θ to maximize $\mathcal{L}_{\text{Bethe}}(\theta, \tilde{q})$ via gradient ascent.

As a result, we see that the SPN can also be understood as solving the Bethe variational problem in Equation (B.5.3), which acts as a surrogate for the log-likelihood function.

B.6. Details of Experiments

Next, we describe our experimental setup in more details.

Dataset	Task	# Features	# Labels	Training Graphs			Validation Graphs			Test Graphs		
				# Graphs	Avg. # Nodes	Avg. # Edges	# Graphs	Avg. # Nodes	Avg. # Edges	# Graphs	Avg. # Nodes	Avg. # Edges
PPI	ML	50	121	20	2245.3	61318.4	2	3257	99460.0	2	2762	80988.0
Cora*	MC	1433	7	140	5.6	7.0	500	4.9	5.8	1000	4.7	5.3
Citeseer*	MC	3703	6	120	4.0	4.3	500	3.8	4.0	1000	3.8	3.8
Pubmed*	MC	500	3	60	6.0	6.7	500	5.4	5.8	1000	5.6	6.7
DBLP	MC	100	3	1	6488	10262	1	14142	48631	1	26813	155899

Table 34. Statistics of datasets used in SPN.

Datasets. The statistics of the datasets used in our experiment are summarized in Table 34. For the Cora*, Citeseer*, Pubmed*, and PPI datasets, they are under the MIT license.

For the DBLP dataset, it is constructed from the citation network [1] in Tang et al. [126]. Scientific papers from eight conferences are treated as nodes, which are divided into three categories based on conference domains [2] for classification. For each paper, we compute the mean GloVe embedding [3] [96] of words in the title and abstract as features. We split the dataset into three disjoint graphs for training/validation/test. The training graph contains papers published before 1999 (with 1999 included). The validation graph contains papers published between 2000 and 2009 (with 2000 and 2009 included). The test graph contains papers published after 2010 (with 2010 included). There exists an undirected edge between two papers if one cites the other one. Cross-split edges (e.g., an edge between a paper in the training set and a paper in the validation set) are removed.

For the PPI datasets, there are 121 binary labels, and we treat each binary label as an independent task. For each compared algorithm, we train a separate model for each task, and report the overall results across all tasks.

Architecture Choices. To facilitate reproducibility, we use the GNN module implementations of PyTorch Geometric [29], and follow the GNN models provided in the examples of the repository, unless otherwise mentioned. Note that most architecture choices are not

[1] `https://originalstatic.aminer.cn/misc/dblp.v12.7z`

[2] ML: ICML/NeurIPS. CV: ICCV/CVPR/ECCV. NLP: ACL/EMNLP/NAACL.

[3] `http://nlp.stanford.edu/data/glove.6B.zip`

optimal on the benchmark datasets, but we did not tune them since we only aim to show that our method brings consistent and significant improvement.

- **GCN [61].** We set the number of hidden neurons to 16, and the number of layers to 2. ReLU [87] is used as the activation function. We do not dropout between GNN layers.
- **GraphSage [46].** We set the number of hidden neurons to 64, and the number of layers to 2. ReLU [87] is used as activation functions. We do not dropout between GNN layers.
- **GAT [137].** We set the number of hidden neurons to 256 per attention head, and the number of layers to 3. The number of heads for each layer is set to 4, 4 and 6. ELU [17] is used as the activation function. We do not dropout between GNN layers.
- **Graph U-Net [33].** We set the number of hidden neurons to 64 and the number of layers to 3. We randomly dropout 20% of the edges from the adjacency matrix. We do not dropout node features or between layers.
- **GCNII [12].** We set the number of hidden neurons to 2048 for the citation datasets (Cora*, Citeseer*, Pubmed* and DBLP) and 256 for the PPI dataset. We set the number of layers to 9. ReLU [87] is used as the activation function. For PPI, layer normalization [1] is applied between the GCNII layers. We do not dropout between GNN layers. We set the strength α of the initial residual connection to 0.5, and the hyperparameter θ to compute the strength of the identity mapping to 1.
- **The g function.** In Equation (4.4.4) of the model section, we define g as a function mapping a pair of L-dimensional representations to a $(|\mathcal{Y}| \times |\mathcal{Y}|)$-dimensional logit. Two variants of this function are used in our experiment. For the PPI and DBLP dataset, we use the linear variant, where the pair of node representations are concatenated and plugged into a linear layer:

$$g_{\text{linear}}(\mathbf{v}_s, \mathbf{v}_t) = \mathbf{W}[\mathbf{v}_s; \mathbf{v}_t] + b, \tag{B.6.1}$$

  where $\mathbf{W} \in \mathbb{R}^{(|\mathcal{Y}|\times|\mathcal{Y}|)\times 2L}$ is the weight matrix and $b \in \mathbb{R}^{|\mathcal{Y}|\times|\mathcal{Y}|}$ is the bias. For the citation datasets (Cora*, Citeseer*, Pubmed*), we use the bilienar variant, where the pair of node representations are plugged in a bilinear mapping:

$$g_{\text{bilinear}}(\mathbf{v}_s, \mathbf{v}_t) = (\mathbf{W}\mathbf{v}_s)(\mathbf{W}\mathbf{v}_t)^T, \tag{B.6.2}$$

  where $\mathbf{W} \in \mathbb{R}^{|\mathcal{Y}|\times L}$ is a weight matrix.
- **SPN with a shared GNN.** By default, the SPN uses a node GNN and an edge GNN to approximate the pseudomarginals on nodes and edges respectively. In the experiment, we also consider using a shared GNN for both pseudomarginals on nodes and edges. In other words, $\mathbf{u}_s = \mathbf{v}_s, \forall s \in V$ (see Equation (4.4.3) and Equation (4.4.4)). All the other components are the same as the default SPN. The results of this variant are shown in Table 16 of the experiment section.

Hyperparameter Choices. For each method, we choose the hyperparameters based on the performance on the validation set. The values of the hyperparameters are shown below.

- **GNNs and SPNs.** For node classification, the learning rate of the node GNN τ_s in GNNs and SPNs is presented in Table 35. For edge classification, the learning rate of the edge GNN τ_{st} is presented in Table 36. For the temperature γ used in the edge GNN τ_{st} of SPNs, we report its values in Table 37.
- **CRF-linear.** For CRF-linear training, we set the learning rate to 5×10^{-4}.
- **CRF-GNNs and SPN.** For CRF and the refinement stage of SPN, we set learning rates to 1×10^{-5}.
- **GMNN.** For GMNN training, we set the learning rate to 5×10^{-3}.

Algorithm	PPI	Cora*	Citeseer*	Pubmed*	DBLP
GCN	5×10^{-3}	5×10^{-3}	1×10^{-2}	1×10^{-2}	1×10^{-2}
GraphSage	5×10^{-3}	5×10^{-3}	5×10^{-3}	5×10^{-3}	5×10^{-3}
GAT	5×10^{-3}	1×10^{-2}	1×10^{-3}	1×10^{-3}	1×10^{-3}
Graph U-Net	5×10^{-3}	1×10^{-2}	1×10^{-2}	1×10^{-2}	-
GCNII	5×10^{-3}	1×10^{-2}	1×10^{-2}	1×10^{-2}	1×10^{-3}

Table 35. Learning rate of the node GNN τ_s in SPN.

Algorithm	PPI	Cora*	Citeseer*	Pubmed*	DBLP
GCN	1×10^{-3}	1×10^{-2}	5×10^{-2}	1×10^{-2}	5×10^{-3}
GraphSage	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-3}
GAT	1×10^{-3}	1×10^{-3}	5×10^{-4}	5×10^{-4}	5×10^{-4}
Graph U-Net	1×10^{-3}	1×10^{-2}	1×10^{-2}	1×10^{-2}	-
GCNII	1×10^{-3}	5×10^{-3}	1×10^{-3}	1×10^{-3}	1×10^{-4}

Table 36. Learning rate of the edge GNN τ_{st} in SPN.

Algorithm	PPI	Cora*	Citeseer*	Pubmed*	DBLP
GCN	10	0.2	1	2	2
GraphSage	10	10	10	10	10
GAT	10	0.2	10	0.2	0.2
Graph U-Net	10	0.5	1	0.2	-
GCNII	10	0.5	0.5	0.5	2

Table 37. Temperature γ of the edge GNN τ_{st} in SPN.

Computational Resources. We run the experiment by using NVIDIA Tesla V100 GPUs with 16GB memory.

Chapter C

pLogicNet: Probabilistic Logic Neural Nets

C.1. Detailed Experiment Settings

In pLogicNet, we parameterize the variational distribution q_θ as a TransE model [8], and we use the method as used in [121] for training the model. More specifically, we define $q_\theta(\mathbf{v}_{(h,r,t)})$ by using a distance-based formulation, i.e., $q_\theta(\mathbf{v}_{(h,r,t)} = 1) = \sigma(\gamma - ||\mathbf{x}_h + \mathbf{x}_r - \mathbf{x}_t||)$, where σ is the sigmoid function and γ is a hyperparameter, which is fixed during training. We generate negative samples by using self-adversarial negative sampling [121], and use Adam [58] as the optimizer. In addition, we filter out unreliable rules and triplets based on the threshold τ_{rule} and τ_{triplet} respectively. During prediction, λ is used to control the relative weight of the knowledge graph embedding model q_θ and the rule-based model p_w. The detailed hyperparameter settings can be found in Table 38.

Dataset	**Embedding Dim.**	**Batch Size**	**# Negative Samples**	α	γ	**Learning Rate**	τ_{rule}	τ_{triplet}	λ
FB15k	1000	2048	128	1.0	24	0.0001	0.1 or 0.8 [a]	0.7	0.5
WN18	500	512	1024	0.5	12	0.0001	0.1	0.5	100
FB15k-237	1000	1024	256	1.0	9	0.00005	0.6	0.7	0.5
WN18RR	500	512	1024	0.5	6	0.00005	0.1	0.5	100

Table 38. Hyperparameters of pLogicNet in the experiment.

[a] For inverse rules and symmetric rules, the threshold is 0.1, whereas for composition rules and subrelation rules, the threshold is 0.8.

Chapter D

RNNLogic: Learning Logic Rules

This section presents the proofs of some propositions used in the optimization algorithm of RNNLogic.

Recall that in the E-step of the optimization algorithm, we aim to sample from the posterior distribution over rule sets. However, directly sampling from the posterior distribution is intractable due to the intractable partition function. Therefore, we introduce Proposition 1, which gives an approximation distribution with more tractable form for the posterior distribution. With this approximation, sampling becomes much easier. In Section D.1, we present the proof of Proposition 2. In Section D.2, we show how to perform sampling based on the approximation of the posterior distribution.

D.1. Proof of Proposition 2 in RNNLogic

Next, we prove Proposition 2, which is used to approximate the true posterior probability in the E-step of optimization. We first restate the proposition as follows:

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

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

where $\mathcal{A}$ *is the set of all candidate answers discovered by rules in* $\hat{\boldsymbol{z}}$*,* $\texttt{score}_w(e|rule)$ *is the score that each rule contributes to entity* e *as defined by Equation* (6.4.4)*,* $\mathrm{RNN}_\theta(rule|r)$ *is the prior probability of rule computed by the generator. Suppose* $s = \max_{e \in \mathcal{A}} |\texttt{score}_w(e)| < 1$*. Then for a subset of rules* $\boldsymbol{z}_I \subset \hat{\boldsymbol{z}}$ *with* $|\boldsymbol{z}_I| = K$*, the log-probability* $\log p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$ *could be approximated as follows:*

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

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

Proof: We first rewrite the posterior probability as follows:

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

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

$$\text{Mu}(\boldsymbol{z}_I|K,\boldsymbol{q}) = \frac{K!}{\prod_{rule\in\hat{\boldsymbol{z}}} n_{rule}!}\prod_{rule\in\hat{\boldsymbol{z}}}\text{RNN}_\theta(rule|r)^{n_{rule}},$$

where K is the size of set $\boldsymbol{z}_I$ and n_{rule} is the number of times a *rule* appears in $\boldsymbol{z}_I$.

Letting $\gamma(\boldsymbol{z}_I) = \log(K!/\prod_{rule\in\hat{\boldsymbol{z}}} n_{rule}!)$, then the posterior probability can then be rewritten as:

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

The above term $\log\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))$ makes the posterior distribution hard to deal with, and 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 $|\texttt{score}_w(e)| \le s < 1$, and let $\texttt{score}_w$ be a function from entities to real numbers. Then the following inequalities hold:

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

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

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

and also the following lower bound of the posterior probability:

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

where const is a constant term which does not depend on $\boldsymbol{z}_I$.

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

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

Thus, it only remains to prove Lemma 1 to complete the proof. We use Theorem D.1.1 from [118] as a starting point:

Theorem D.1.1. *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 convex and $\exp(\texttt{score}_w(e)) \in [\exp(-s), \exp(s)]$, Theorem D.1.1 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:

$$
\begin{aligned}
&\log\left(\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right) \\
&\leq \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e) + \log(|\mathcal{A}|) + 2\log\left(\frac{\exp(-s)+\exp(s)}{2}\right) \\
&= \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e) + \log(|\mathcal{A}|) + 2s - 2\log 2 + 2\log(1+\exp(-2s)) \\
&\leq \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e) + \log(|\mathcal{A}|) + s^2 + O(s^4),
\end{aligned}
\tag{D.1.1}
$$

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:

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

which implies:

$$\log\left(\sum_{e\in\mathcal{A}}\exp(\texttt{score}_w(e))\right) \geq \sum_{e\in\mathcal{A}}\frac{1}{|\mathcal{A}|}\texttt{score}_w(e) + \log(|\mathcal{A}|). \tag{D.1.2}$$

By combining Equation (D.1.1) and Equation (D.1.2), we obtain:

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

This completes the proof.

□.

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

Based on Proposition 1, the log-posterior probability $\log p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G},\boldsymbol{q},\boldsymbol{a})$ could be approximated by $(\sum_{rule\in\boldsymbol{z}_I} H(rule)+\gamma(\boldsymbol{z}_I)+\text{const})$, with const being a term that does not depend on $\boldsymbol{z}_I$. This implies that we could construct a distribution $q(\boldsymbol{z}_I) \propto \exp(\sum_{rule\in\boldsymbol{z}_I} H(rule)+\gamma(\boldsymbol{z}_I))$ to approximate the true posterior, and draw samples from q as approximation to the real samples from the posterior.

It turns out that the distribution $q(\boldsymbol{z}_I)$ is a multinomial distribution. To see that, we rewrite $q(\boldsymbol{z}_I)$ as:

$$\begin{aligned}
q(\boldsymbol{z}_I) &= \frac{1}{Z}\exp\left(\sum_{rule\in\boldsymbol{z}_I} H(rule) + \gamma(\boldsymbol{z}_I)\right) \\
&= \frac{1}{Z}\exp\left(\gamma(\boldsymbol{z}_I)\right)\prod_{rule\in\boldsymbol{z}_I}\exp\left(H(rule)\right) \\
&= \frac{1}{Z}\frac{K!}{\prod_{rule\in\hat{\boldsymbol{z}}} n_{rule}!}\prod_{rule\in\hat{\boldsymbol{z}}}\exp\left(H(rule)\right)^{n_{rule}} \\
&= \frac{1}{Z'}\frac{K!}{\prod_{rule\in\hat{\boldsymbol{z}}} n_{rule}!}\prod_{rule\in\hat{\boldsymbol{z}}} q_r(rule)^{n_{rule}} \\
&= \frac{1}{Z'}\text{Mu}(\boldsymbol{z}_I|K,q_r),
\end{aligned}$$

where n_{rule} is the number of times a *rule* appears in the set $\boldsymbol{z}_I$, q_r is a distribution over all the generated logic rules $\hat{\boldsymbol{z}}$ with $q_r(rule) = \exp(H(rule))/\sum_{rule'\in\hat{\boldsymbol{z}}}\exp(H(rule'))$, Z and Z' are

normalization terms. By summing over $\boldsymbol{z}_I$ on both sides of the above equation, we obtain $Z' = 1$, and hence:

$$q(\boldsymbol{z}_I) = \mathrm{Mu}(\boldsymbol{z}_I|K, q_r).$$

To sample from such a multinomial distribution, we could simply sample K rules independently from the distribution q_r, and form a sample $\hat{\boldsymbol{z}}_I$ with these K rules.

In practice, we observe that the hard-assignment EM algorithm [64] works better than the standard EM algorithm despite the reduced theoretical guarantees. In the hard-assignment EM algorithm, we need to draw a sample $\hat{\boldsymbol{z}}_I$ with the maximum posterior probability. Based on the above approximation $q(\boldsymbol{z}_I)$ of the true posterior distribution $p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$, we could simply construct such a sample $\hat{\boldsymbol{z}}_I$ with K rules which have the maximum probability under the distribution q_r. By definition, we have $q_r(rule) \propto \exp(H(rule))$, and hence drawing K rules with maximum probability under q_r is equivalent to choosing K rules with the maximum H values.

D.3. More Analysis of the EM Algorithm

In RNNLogic, we use an EM algorithm to optimize the rule generator. In this section, we show why this EM algorithm is able to maximize the objective function of the rule generator.

Recall that for a fixed reasoning predictor p_w, we aim to update p_θ to maximize the log-likelihood function $\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q})$ for each data instance $(\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$. Directly optimizing $\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q})$ is difficult due to the latent logic rules, and therefore we consider the following evidence lower bound of the log-likelihood function:

$$\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q}) \geq \mathbb{E}_{q(\boldsymbol{z}_I)}[\log p_w(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q}, \boldsymbol{z}_I) + \log p_\theta(\boldsymbol{z}_I|\boldsymbol{q}) - \log q(\boldsymbol{z}_I)] = \mathcal{L}_{\mathrm{ELBO}}(q, p_\theta), \tag{D.3.1}$$

where $q(\boldsymbol{z}_I)$ is a variational distribution, and the equation holds when $q(\boldsymbol{z}_I) = p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$.

With this lower bound, we can optimize the log-likelihood function $\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q})$ with an E-step and an M-step. In the E-step, we optimize $q(\boldsymbol{z}_I)$ to maximize $\mathcal{L}_{\mathrm{ELBO}}(q, p_\theta)$, which is equivalent to minimizing $\mathrm{KL}(q(\boldsymbol{z}_I)||p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a}))$. By doing so, we are able to tighten the lower bound. Then in the M-step, we further optimize θ to maximize $\mathcal{L}_{\mathrm{ELBO}}(q, p_\theta)$. Next, we introduce the details.

E-step. In the E-step, our goal is to update q to minimize $\mathrm{KL}(q(\boldsymbol{z}_I)||p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a}))$. However, there are a huge number of possible logic rules, and hence optimizing $q(\boldsymbol{z}_I)$ on every possible rule set $\boldsymbol{z}_I$ is intractable. To solve the problem, recall that we generate a set of logic rules $\hat{\boldsymbol{z}}$ when optimizing the reasoning predictor, and here we add a constraint to q based on $\hat{\boldsymbol{z}}$. Specifically, we constrain the sample space of $q(\boldsymbol{z}_I)$ to be all subsets of $\hat{\boldsymbol{z}}$ with size

being K, i.e., $\boldsymbol{z}_I \subset \hat{\boldsymbol{z}}$ and $|\boldsymbol{z}_I| = K$. In other words, we require $\sum_{\boldsymbol{z}_I \subset \hat{\boldsymbol{z}}, |\boldsymbol{z}_I|=K} q(\boldsymbol{z}_I) = 1$. With such a constraint, we can further use Proposition 2 to construct the variational distribution q to approximate $p_{\theta,w}(\boldsymbol{z}_I|\mathcal{G}, \boldsymbol{q}, \boldsymbol{a})$, as what is described in the model section.

M-step. In the M-step, our goal is to update p_θ to maximize the lower bound $\mathcal{L}_{\text{ELBO}}(q, p_\theta)$. To do that, we notice that there is an expectation operation with respect to $q(\boldsymbol{z}_I)$ in $\mathcal{L}_{\text{ELBO}}(q, p_\theta)$. By drawing a sample from $q(\boldsymbol{z}_I)$, $\mathcal{L}_{\text{ELBO}}(q, p_\theta)$ can be estimated as follows:

$$\begin{aligned}\mathcal{L}_{\text{ELBO}}(q, p_\theta) &= \mathbb{E}_{q(\boldsymbol{z}_I)}[\log p_w(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q}, \boldsymbol{z}_I) + \log p_\theta(\boldsymbol{z}_I|\boldsymbol{q}) - \log q(\boldsymbol{z}_I)] \\ &\simeq \log p_w(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q}, \hat{\boldsymbol{z}}_I) + \log p_\theta(\hat{\boldsymbol{z}}_I|\boldsymbol{q}) - \log q(\hat{\boldsymbol{z}}_I),\end{aligned} \tag{D.3.2}$$

where $\hat{\boldsymbol{z}}_I \sim q(\boldsymbol{z}_I)$ is a sample drawn from the variational distribution. By ignoring the terms which are irrelevant to p_θ, we obtain the following objective function for θ:

$$\log p_\theta(\hat{\boldsymbol{z}}_I|\boldsymbol{q}), \tag{D.3.3}$$

which is the same as the objective function described in the model section.

As a result, by performing the E-step and the M-step described in the model section, we are able to update p_θ to increase the lower bound $\mathcal{L}_{\text{ELBO}}(q, p_\theta)$, and thereby push up the log-likelihood function $\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q})$. Therefore, we see that the EM algorithm can indeed maximize $\log p_{w,\theta}(\boldsymbol{a}|\mathcal{G}, \boldsymbol{q})$ with respect to p_θ.

D.4. Details of Parameterization and Implementation

Section 6.4.2 of the paper introduces the high-level idea of the reasoning predictor with logic rules and the rule generator. Due to the limited space, some details of the models are not covered. In this section, we explain the details of the reasoning predictor and the rule generator.

Reasoning Predictor with Logic Rules. We start with the reasoning predictor with logic rules. Recall that for each query, our reasoning predictor leverages a set of logic rules $\boldsymbol{z}$ to give each candidate answer a score, which is further used to predict the correct answer from all candidates.

Specifically, let $\mathcal{A}$ denote the set of all the candidate answers discovered by logic rules in set $\boldsymbol{z}$. For each candidate answer $e \in \mathcal{A}$, we define the following function $\texttt{score}_w$ to compute a score:

$$\texttt{score}_w(e) = \sum_{rule \in \boldsymbol{z}} \texttt{score}_w(e|rule) = \sum_{rule \in \boldsymbol{z}} \sum_{path \in \mathcal{P}(h,rule,e)} \psi_w(rule) \cdot \phi_w(path), \tag{D.4.1}$$

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

For the scalar weight $\psi_w(rule)$ of a *rule*, we initialize $\psi_w(rule)$ as follows:

$$\psi_w(rule) = \mathbb{E}_{(\mathcal{G},\boldsymbol{q},\boldsymbol{a})\sim p_{\text{data}}}\left[|\mathcal{P}(h,rule,t)| - \frac{1}{|\mathcal{A}|}\sum_{e\in\mathcal{A}}|\mathcal{P}(h,rule,e)|\right], \tag{D.4.2}$$

where $|\mathcal{P}(h,rule,t)|$ is the number of relational paths starting from head entity h, following the relations in *rule* and ending at tail entity t. The form is very similar to the definition of H values for logic rules, and the value can effectively measure the contribution of a rule to the correct answers. We also try randomly initializing rule weights or initializing them as 0, which yield similar results.

For the scalar score $\phi_w(path)$ of a *path*, we either fix it to 1, or compute it by introducing entity and relation embeddings. In the second case, we introduce an embedding for each entity and relation in the complex space. Formally, the embedding of an entity e is denoted as $\boldsymbol{x}_e$, and the embedding of a relation $\mathtt{r}$ is denoted as $\boldsymbol{x}_{\mathtt{r}}$. For a grounding path $path = e_0 \xrightarrow{\mathtt{r}_1} e_1 \xrightarrow{\mathtt{r}_2} e_2 \cdots e_{l-1} \xrightarrow{\mathtt{r}_l} e_l$, we follow the idea in RotatE [121] and compute $\phi_w(path)$ in the following way:

$$\phi_w(path) = \sigma(\delta - d(\boldsymbol{x}_{e_0} \circ \boldsymbol{x}_{\mathtt{r}_1} \circ \boldsymbol{x}_{\mathtt{r}_2} \circ \cdots \circ \boldsymbol{x}_{\mathtt{r}_l}, \boldsymbol{x}_{e_l})), \tag{D.4.3}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function, d is a distance function between two complex vectors, δ is a hyperparameter, and $\circ$ is the Hadmard product in complex spaces, which could be viewed as a rotation operator. Intuitively, for the embedding $\boldsymbol{x}_{e_0}$ of entity e_0, we rotate $\boldsymbol{x}_{e_0}$ by using the rotation operators defined by $\{\mathtt{r}_k\}_{k=1}^{l}$, yielding $(\boldsymbol{x}_{e_0} \circ \boldsymbol{x}_{\mathtt{r}_1} \circ \boldsymbol{x}_{\mathtt{r}_2} \circ \cdots \circ \boldsymbol{x}_{\mathtt{r}_l})$. Then we compute the distance between the new embedding and the embedding $\boldsymbol{x}_{e_l}$ of entity e_l, and further convert the distance to a value between 0 and 1 by using the sigmoid function and a hyperparameter δ.

Rule Generator. This paper focuses on compositional rules, which have the abbreviation form $\mathtt{r} \leftarrow \mathtt{r}_1 \wedge \cdots \wedge \mathtt{r}_l$ and thus could be viewed a sequence of 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. We introduce a rule generator RNN_θ parameterized with an LSTM [49] to model such sequences. Given the current relation sequence $[\mathtt{r}, \mathtt{r}_1, \mathtt{r}_2 \cdots \mathtt{r}_i]$, RNN_θ aims to generate the next relation $\mathtt{r}_{i+1}$ and meanwhile output the probability of $\mathtt{r}_{i+1}$. The detailed computational process towards the goal is summarized as follows:

- Initialize the hidden state of the RNN_θ as follows:

$$\boldsymbol{h}_0 = f(\boldsymbol{v}_{\mathtt{r}}),$$

  where $\boldsymbol{v}_{\mathtt{r}}$ is a parameter vector associated with the query relation or the head relation $\mathtt{r}$, f is a linear transformation.

- Sequentially compute the hidden state at different positions by using the LSTM gate:

$$\boldsymbol{h}_t = \text{LSTM}(\boldsymbol{h}_{t-1}, g([\boldsymbol{v}_{\mathtt{r}}, \boldsymbol{v}_{\mathtt{r}_t}]),$$

  where $\boldsymbol{v}_{\mathtt{r}_t}$ is a parameter vector associated with the relation $\mathtt{r}_t$, $[\boldsymbol{v}_{\mathtt{r}}, \boldsymbol{v}_{\mathtt{r}_t}]$ is the concatenation of $\boldsymbol{v}_{\mathtt{r}}$ and $\boldsymbol{v}_{\mathtt{r}_t}$, g is a linear transformation.
- Generate $\mathtt{r}_{t+1}$ and its probability based on $\boldsymbol{h}_{t+1}$ and the following vector:

$$\text{softmax}(o(\boldsymbol{h}_{t+1})).$$

  Suppose the set of relations is denoted as $\mathcal{R}$. We first transform $\boldsymbol{h}_{t+1}$ to a $|\mathcal{R}|$-dimensional vector by using a linear transformation o, and then apply softmax function to the $|\mathcal{R}|$-dimensional vector to get the probability of each relation. Finally, we generate $\mathtt{r}_{t+1}$ according to the probability vector.

D.5. Details of Experiments

The statistics of datasets are summarised in Table 39.

Dataset	#Entities	#Relations	#Train	#Validation	#Test
FB15K-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
Kinship	104	25	3,206	2,137	5,343
UMLS	135	46	1,959	1,306	3,264

Table 39. Statistics of datasets used in RNNLogic.

Next, we explain the detailed experimental setup of RNNLogic. We try different configurations of hyperparameters on the validation set, and the optimal configuration is then used for testing. We report the optimal hyperparameter configuration as below.

Data Preprocessing. For each training triplet $(h, \mathtt{r}, t)$, we add an inverse triplet $(t, \mathtt{r}^{-1}, h)$ into the training set, yielding an augmented set of training triplets $\mathcal{T}$. To build a training instance from p_{data}, we first randomly sample a triplet $(h, \mathtt{r}, t)$ from $\mathcal{T}$, and then form an instance as $(\mathcal{G} = \mathcal{T} \setminus \{(h, \mathtt{r}, t)\}, \boldsymbol{q} = (h, \mathtt{r}, ?), \boldsymbol{a} = t)$. Basically, we use the sampled triplet $(h, \mathtt{r}, t)$ to construct the query and answer, and use the rest of triplets in $\mathcal{T}$ to form the background knowledge graph $\mathcal{G}$. During testing, the background knowledge graph $\mathcal{G}$ is constructed by using all the triplets in $\mathcal{T}$.

Reasoning Predictor. For the reasoning predictor in *with embedding* cases, the embedding dimension is set to 500 for FB15k-237, 200 for WN18RR, 2000 for Kinship and 1000 for UMLS. We pre-train these embeddings with RotatE [121]. The hyperparameter δ for computing $\phi_w(path)$ in Equation (D.4.3) is set to 9 for FB15k-237, 6 for WN18RR, 0.25 for

Kinship and 3 for UMLS. We use the Adam [58] optimizer with an initial learning rate being 5×10^{-5}, and we decrease the learning rate in a cosine shape.

Rule Generator. For the rule generator, the maximum length of generated rules is set to 4 for FB15k-237, 5 for WN18RR, and 3 for Kinship and UMLS. These numbers are chosen according to model performace on the validation data. The size of input and hidden states in RNN_θ are set to 512 and 256. The learning rate is set to 1×10^{-3} and monotonically decreased in a cosine shape. Beam search is used to generate rules with high probabilities, so that we focus on exploiting these logic rules which the rule generator is confident about. Besides, we pre-train the rule generator by using sampled relational paths on the background knowledge graph formed with training triplets, which prevents the rule generator from exploring meaningless logic rules in the beginning of training.

EM Optimization. During optimization, we sample 1000 rules from the rule generator for each data instance. In the E-step, for each data instance, we identify 300 rules as high-quality logic rules.

Evaluation. In testing, for each query $\boldsymbol{q} = (h, r, ?)$, we use the rule generator p_θ to generate 1000 logic rules, and let the reasoning predictor p_w use the generated logic rules to predict the answer $\boldsymbol{a}$.

Chapter E

DiffLogic: A Differentiable Approach of Rule Learning

This section presents the proofs of claims used in DiffLogic.

E.1. Derivation of the Gradient

The the model section, we have used the following formular regarding the gradient of the log-likelihood function with respect to θ and ϕ:

$$\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) = \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})], \tag{E.1.1}$$

$$\nabla_\phi \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) = \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\phi \log p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z})]. \tag{E.1.2}$$

Next, we prove the above formular.

We first consider θ, and we have:

$$\begin{aligned}
\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) &= \frac{\nabla_\theta \sum_{\boldsymbol{z}} p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} = \frac{\sum_{\boldsymbol{z}} \nabla_\theta p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} \\
&= \sum_{\boldsymbol{z}} \frac{\nabla_\theta p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} \\
&= \sum_{\boldsymbol{z}} \frac{1}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} \nabla_\theta p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x}) \\
&= \sum_{\boldsymbol{z}} \frac{1}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x}) \nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x}) \\
&= \sum_{\boldsymbol{z}} \frac{p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})}{p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x})} \nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x}) \\
&= \sum_{\boldsymbol{z}} p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y}) \nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x}) \\
&= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})].
\end{aligned} \tag{E.1.3}$$

Based on this, we further have:

$$\begin{aligned}\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})] \\ &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta(\log p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z}) + \log p_\theta(\boldsymbol{z}|\mathtt{r}))] \\ &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z}) + \nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})] \\ &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\theta \log p_\theta(\boldsymbol{z}|\mathtt{r})].\end{aligned} \tag{E.1.4}$$

Similarly, we have the same conclusion for ϕ as below:

$$\begin{aligned}\nabla_\phi \log p_{\phi,\theta}(\boldsymbol{y}|\mathcal{G},\boldsymbol{x}) &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\phi \log p_{\phi,\theta}(\boldsymbol{y},\boldsymbol{z}|\mathcal{G},\boldsymbol{x})] \\ &= \mathbb{E}_{p_{\phi,\theta}(\boldsymbol{z}|\mathcal{G},\boldsymbol{x},\boldsymbol{y})}[\nabla_\phi \log p_\phi(\boldsymbol{y}|\mathcal{G},\boldsymbol{x},\boldsymbol{z})].\end{aligned} \tag{E.1.5}$$

E.2. Details of Experiments

The statistics of datasets are summarised in Table 40.

Dataset	#Entities	#Relations	#Train	#Validation	#Test
FB15K-237	14,541	237	272,115	17,535	20,466
WN18RR	40,943	11	86,835	3,034	3,134
Kinship	104	25	3,206	2,137	5,343
UMLS	135	46	1,959	1,306	3,264

Table 40. Statistics of datasets used in DiffLogic.