Neural, Symbolic and Neural-Symbolic Reasoning on Knowledge Graphs

Jing Zhang*, Bo Chen, Lingxi Zhang, Xirui Ke, Haipeng Ding

Abstract—Knowledge graph reasoning is the fundamental component to support machine learning applications such as information extraction, information retrieval, and recommendation. Since knowledge graphs can be viewed as the discrete symbolic representations of knowledge, reasoning on knowledge graphs can naturally leverage the symbolic techniques. However, symbolic reasoning is intolerant of the ambiguous and noisy data. On the contrary, the recent advances of deep learning promote neural reasoning on knowledge graphs, which is robust to the ambiguous and noisy data, but lacks interpretability compared to symbolic reasoning. Considering the advantages and disadvantages of both methodologies, recent efforts have been made on combining the two reasoning methods. In this survey, we take a thorough look at the development of the symbolic, neural and hybrid reasoning on knowledge graphs. We survey two specific reasoning tasks — knowledge graph completion and question answering on knowledge graphs, and explain them in a unified reasoning framework. We also briefly discuss the future directions for knowledge graph reasoning.

Index Terms—Knowledge Graph Reasoning, Knowledge Graph Embedding, Symbolic Reasoning, Neural-symbolic Reasoning

1 Introduction

Symbolism and connectionism are two main paradigms of Artificial Intelligence. Symbolism assumes the basic units which compose the human intelligence are symbols, and the cognitive process is a series of explicit inferences upon symbolic representations [92], [54]. Generally, the models of symbolism have sound readability and interpretability. However, the finite and discrete symbolic representations are insufficient to depict all the intrinsic relationships among data, and also intolerant of ambiguous and noisy data. On the contrary, connectionism imitates the process of neuron connections and cooperation in human brains to build models [115], [117]. Different models of connectionism have been developed. Deep learning is a representative model of connectionism [4], [58]. Deep learning has reached unprecedented impact across research communities as it achieved superior performances on many tasks in different fields such as image classification in computer vision [16], [56], [57], language modeling in natural language processing [12], [31], and link prediction in networks [72], [141], [166].This indicates that deep learning models are capable of modeling the implicit correlations inside data. However, the models of connectionism cannot provide explicit inference evidence to explain the results, making it look like a black box. The problem of combining symbolism and connectionism has been researched since the 1980s [44], [43], [123], [133], [132], and has attracted much attention recent years [6], [28], [45], [46], [76].

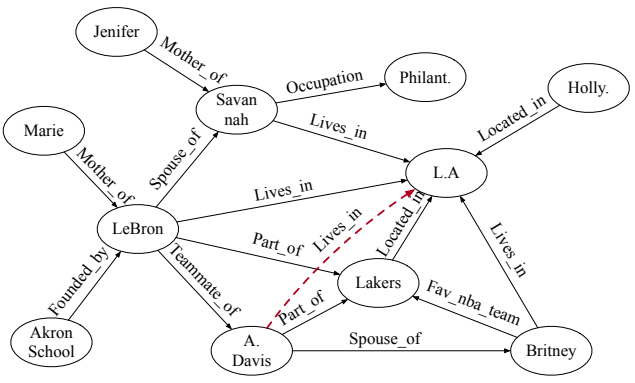


Fig. 1. Illustration of a knowledge graph [130] and examples of two reasoning tasks.

In this survey, we discuss how the above two principles are performed and intertwined for reasoning over knowledge graphs (KGs) such as Freebase [8], DBpedia [80], YAGO [125] and NELL [14], which are composed by a large number of (head, relation, and tail) triplets. Since KGs are naturally discrete symbolic representations to support explicit inferences and can be also represented into the continuous vector space by deep learning models such as the state-of-the-art TransE [10] to support implicit inferences, it is promising to effectively combine the ideas of symbolism and

• *Jing Zhang, Bo Chen, Lingxi Zhang, Xirui Ke, Haipeng Ding are with Information School, Renmin University of China, Beijing, China. E-mail:zhang-jing@ruc.edu.cn, allanchen224@gmail.com, zhanglingxi@ruc.edu.cn, kexirui@ruc.edu.cn, dinghaipeng@ruc.edu.cn,*
**Corresponding author*

connectionsim on KG reasoning tasks. Specifically, we review the mechanisms of two typical KG reasoning tasks — knowledge graph completion (KGC) and knowledge graph question answering (KGQA). KGC is to infer the answers, i.e., the tail entities given a head entity and a query relation [26], [78], [153], [160]. It is formulated as $(h, r, ?)$ where h is the head entity, r is the query relation, and $?$ is the tail entity that should satisfy r with h. KGQA is similar to KGC except that the condition of the head entity h and the query relation r is replaced by a natural language question q [2], [5], [119], [127]. Figure 1 illustrates the two reasoning tasks by examples, where the KGC example is to reason the tail entity "L.A" given the query relation "Lives_in" and the head entity "A. Davis", and the KGQA example is to reason the answer "L.A" given the question "Where do the spouses of the teammates of Lakers usually live?".

Despite the abundant surveys about knowledge graph embedding [67], [116], [145], acquisition and applications [67], question answering [38], [150] and reasoning [18], none of them explicitly sort out the symbolic or the neural methods, or the way to intertwine them. All the aforementioned surveys separate the reasoning tasks of KGC and KGQA. This paper unifies the two tasks in one reasoning framework and categorize the reasoning mechanisms into symbolic, neural, and hybrid. We demonstrate the pros. and cons. of each category and also discuss the future directions for KG reasoning. All the surveyed papers are retrieved by Google Scholar according to the keywords "knowledge graph reasoning", "knowledge graph completion", "knowledge graph question answering", etc., and are chosen to be included if they are published by authoritative conferences or journals or widely cited.

The rest of the article is organized as follows. Section 2 introduces the background knowledge that is closely related to the surveyed techniques. Section 3 reviews the three kinds of reasoning techniques for knowledge graph completion. Section 4 reviews question answering on KGs under a similar reasoning framework. Section 5 first summarizes all kinds of reasoning methods in a unified technique development trend and then ends with the discussion of the potential directions in the future.

2 Background Knowledge and Problem Definition

In this section, we first introduce the background information about knowledge graph, inductive logic programming, and Markov network, and then give the definition of knowledge graph reasoning.

2.1 Knowledge Graphs (KGs)

We denote a knowledge graph G as a set of facts, each represented by a triplet (h, r, t), where h is a head entity, r is a relation and t is a tail entity. In Figure 1, (LeBron, part_of Lakers) is an example of a triplet, where LeBron is the head entity, part_of is the relation and Lakers is the tail entity.

2.2 Inductive Logic Programming (ILP)

Inductive Logic Programming (ILP) aims at seeking underlying patterns formulated by logic programs/rules/formulas shared in the data. It is one of the rule-based learning methods which derive a set of if-then logic rules to describe the positive instances but not the negative instances. Most ILP works constrain the logic rule to be Horn clause/rule. A Horn rule consists of a head and a body, where the head is a single atom and the body is flat conjunction over several atoms. A Horn rule γ can be formulated as:

$$\gamma : A(\alpha_1, \cdots, \alpha_m) \rightarrow \alpha, \tag{1}$$

where α is called head atom and $\alpha_1, \cdots, \alpha_m$ $(m \geq 0)$ are body atoms. A is the rule body which is usually defined as a conjunction normal form (CNF) that uses logical operations $\{\wedge, \vee, \neg\}$ to combine the body atoms together. The rule body can also be referred to as a formula. If the rule body A on the left is true, then the head atom on the right is also true. An atom α is defined as a predicate symbol P_i that acts as a function to map the set of variables $\{x_1, x_2, \cdots, x_n\}$ to true or false:

$$\alpha \equiv P_i(x_1, x_2, \cdots, x_n). \tag{2}$$

Although there can be multiple variables in a predicate, we usually only consider the simple unary and binary predicates, i.e., $n = 1$ and $n = 2$. For example, Person is a unary predicate only applied to a single variable. The atom Person(x) is true if x is a person. Mother is a binary predicate applied to two variables. The atom Mother(x, y) is true if x is the mother of y. When all the variables in an atom α are instantiated by constants, α is called a ground atom.

In a knowledge graph, a relation r can be viewed as a binary predicate; that is, $r(x, y)$ is an atom with two arguments x and y. A triplet $(h, r, t) \in G$ can be taken as a ground atom $r(h, t)$ which applies a relation r to a pair of entities h and t.

Given a set of pre-defined predicates $\mathcal{P}$, ground atoms $\mathcal{G}$, positive instances $\mathcal{S}$ and negative instances $\mathcal{N}$, ILP aims at constructing a set of rules to explain the positive instances and reject the negative instances. We take the example of learning which natural numbers are even from [36] to explain ILP. The predicate set is defined as:

$$\mathcal{P} = \{\text{zero}, \text{succ}\},$$

where zero(X) is an unary predicate which is true if X is 0, succ(X, Y) is a binary predicate which is true if X is the successor of Y. The ground atoms are:

$$\mathcal{G} = \{\text{zero}(0), \text{succ}(0,1), \text{succ}(1,2), \cdots\}.$$

The positive and negative instances of the even predicate are:

$$\begin{aligned} \mathcal{S} &= \{\text{even}(0), \text{even}(2), \text{even}(4), \cdots\}, \\ \mathcal{N} &= \{\text{even}(1), \text{even}(3), \text{even}(5), \cdots\}. \end{aligned}$$

The solution of rules for the even predicate is:

$$\begin{aligned} \text{even}(X) &\leftarrow \text{zero}(X), \\ \text{even}(X) &\leftarrow \text{even}(Y) \wedge \text{succ2}(Y, X), \\ \text{succ2}(X, Y) &\leftarrow \text{succ}(X, Z) \wedge \text{succ}(Z, Y). \end{aligned}$$

where succ2(X, Y) is an auxiliary predicate which is true when X is the two-hop successor of Y. We can see that when the above rules are applied deductively to the ground atoms $\mathcal{G}$, they can produce $\mathcal{S}$ but not $\mathcal{N}$.

Two kinds of approaches are usually used to derive rules, where the top-down approach begins from general rules and adds new atoms to improve the coverage precision of positive instances [21], [91], while the bottom-up approach begins from the specific rules and deletes atoms to extend the coverage rate of the rules [106], [107].

2.3 Markov Network

Markov network, also known as Markov random field, tries to model the knowledge graph using the joint distribution of a set of variables $X = (X_1, X_2, ..., X_n)$ [104]. Markov network is an undirected graph where each node represents a variable. For cliques in the graph, a nonnegative real-valued potential function ϕ_c is defined and the joint distribution is represented as:

$$P(X = x) = \frac{1}{Z} \prod_c \phi_c(x_c) \tag{3}$$

where x_c reflects the state of the variables appearing in the c-th clique. $Z = \sum_{x \in \mathcal{X}} \prod_c \phi_c(x_c)$ is the partition function for normalization. Markov networks are usually represented as log-linear models, with each clique potential replaced by an exponentiated weighted sum of the state's feature:

$$P(X = x) = \frac{1}{Z} \exp(\sum_c w_c f_c(x_c)) \tag{4}$$

where f_c is the feature function defined on the clique c and w_c is the corresponding vector of weights.

3 Reasoning for KGC

This section introduces the reasoning methods for KGC in three main categories. Figure **??** presents the taxonomy of the KGC reasoning methods.

3.1 Neural Reasoning

Neural reasoning, also known as knowledge graph embedding, aims at learning the distributed embeddings for entities and relations in KGs and inferring the answer entities based on embeddings when given the head entity and relation. Generally, existing neural reasoning methods can be categorized into translation-based models, multiplicative models, and deep learning models.

3.1.1 Translation-based Models

Translation-based models usually learn embeddings by translating a head entity to a tail entity through the relation. For example, TransE [10], a representative translation-based model, maps the entities and relations into the same vector space and forces the added embedding $\mathbf{h}+\mathbf{r}$ of a head entity h and a relation r to be close to the embedding $\mathbf{t}$ of the corresponding tail entity t, i.e., minimizes the score of a triple as follows:

$$s(h, r, t) = ||\mathbf{h} + \mathbf{r} - \mathbf{t}||_2^2. \tag{5}$$

Subsequently, various models have been proposed to improve the capability of TransE. For example, TransH deals with the flaws caused by the relations with the properties of reflexive, one-to-many, many-to-one, and many-to-many by projecting the entities into the relation-specific hyperplane, which enables different roles of an entity in different relations/triplets [148]. Instead of projecting entities and relations into the same space, TransR builds entity and relation embeddings in separate entity space and relation space [84]. TransD [66] determines the mapping matrices by both the entities and the relations with the hope to capture the diversity of entities and relations simultaneously. TransG [151] further addresses the ambiguity of a relation by incorporating a Bayesian non-parametric infinite mixture model to generate multiple translation components for a relation. TransAt [108] determines a relation between two entities in two steps inspired by the human cognitive process. It first checks the categories of entities and then determines a specific relation by relation-related attributes through a relation-aware attention mechanism. To model and infer the symmetry/antisymmetry, inversion, and composition patterns together, RotatE [129] maps the entities and relations to the complex vector space and defines each relation as a rotation from the head entity to the target entity.

TABLE 1
A summary of knowledge graph completion and recent advances.

Category	Sub-category	Model	Mechanism
Neural Reasoning	Translation -based Models	TransE [10]	force $\mathbf{h}+\mathbf{r}$ to be close to $\mathbf{t}$
		TransH [148]	project entities into the relation-specific hyperplane
		TransR [84]	project entities and relations in separate spaces
		TransD [66]	determine the project matrices by both entities and relations
		TransG [151]	project a relation into multiple embeddings
		TransAt [108]	project a relation by entities' categories and the relation's attributes
		RotatE [129]	map entities and relations to the complex vector space
	Multiplicative Models	RESCAL [99]	maximize the tensor product between two entities
		DisMult [159]	simplify the project matrix into a diagonal matrix
		ComplEx [134]	use complex embeddings to handle asymmetry
		HolE [98]	use the circular correlation to represent entity pairs
		SimplE [70]	learn a head and tail embeddings for each entity
		ANALOGY [85]	constrain relation matrixes to be normal matrixes
		DiHEdral [155]	support the non-commutative property of a relation
	Deep Learning Models	ConvE [30]	use the head and relation embeddings as the input of CNN
		ConvR [68]	extend global filters in ConvE to relation-specific filters
		ConvKB [97]	use the head, relation and tail together as the input of CNN
		CapsE [142]	use the capsule network as the convolution function
		RSN [50]	use RNNs to capture long-term relational dependencies
		R-GCN [120]	use relation-aware GCN as the encoder and DisMult as the decoder
		M-GNN [147]	replace the mean aggregator in R-GCN with a MLP
		CompGCN [140]	incorporate entity and relation embeddings into the aggregator function
		KBGAT [94]	incorporate a triplet into the aggregator function
Symbolic Reasoning	-	AIME [42]	generate rules by rule extending and rule pruning
		AIME+ [41]	improve the efficiency of AMIE
		RLvLR [102]	reduce the search space by the embedding technique
		RuLES [60]	leverage the embedding technique to measure the quality of the rules
Neural-Symbolic Reasoning	Symbolic-driven Neural Reasoning	KALE [51]	learn embeddings on the observed triples and the ground rules
		RUGE [52]	change the ground rules to the triplets derived by rules
		Wang et al. [143]	transform a triple/ground rule into first-order logic
		IterE [168]	infer rules and update embeddings iteratively
	Symbolic-driven Probabilistic Reasoning	MLN [113]	build a probabilistic graphic model for all the rules and learn their weights
		pLogicNet [111]	incorporate the embedding technique to infer marginal probabilities in MLN
		ProbLog [29]	build a local SLD-tree for a relation and learn rules that can support it
		SLPs [24]	define a randomized procedure for traversing the SLD-tree in ProbLog
		ProPPR [146]	change the randomized procedure in SLPs to a biased sampling strategy
	Neural-driven Symbolic Reasoning	PRA [78]	enumerate the paths between two entities
		Lao et al. [79]	perform PRA on the KG and the extended textural graph
		Neelakantan et al. [96]	use RNN to encode the most confident path
		Chain-of-Reasoning [27]	change the most confident path to multiple paths
		DeepPath [153]	use RL to evaluate the sampled paths
		AnyBURL [90]	generalize the sampled paths to abstract rules
		MINERVA [26]	use RL to directly find the answer
		MultiHop [83]	adopt soft reward and action dropout for MINERVA
		CPL [39]	leverage the text in addition to the KG when sampling
		M-walk [124]	use Monte Carlo Tree Search when sampling
		DIVA [17]	use VAE to unify path sampling and answer reasoning
		CogGraph [34]	sample multiple entities at each hop
		TensorLog [22]	keep all the neighbors at each hop without sampling
		Neural LP [160]	learn new rules based on TensorLog
		NLIL [162]	deal with non-chain-like rules
		Neural-Num-LP [144]	deal with numerical operations

3.1.2 Multiplicative Models

Multiplicative models produce the entity and relation embeddings via tensor product as follows:

$$s(h,r,t)=\mathbf{h}^T\mathbf{M}_r\mathbf{t}, \tag{6}$$

where $\mathbf{M}_r$ is an asymmetric $d\times d$ matrix that models the interactions of the latent components in the r-th relation. Given a relation matrix M_r, a feature in the above tensor product is "on" if and only if the corresponding features of both entities h and t are "on", which can capture the relational patterns between entities. The representative tensor product-based models are RESCAL [99]. However, the tensor product requires a large number of parameters as it models all pairwise interactions. To reduce the computational cost, DisMult [159] is proposed to use the diagonal matrix with the diagonal vector indicating the embedding of the relation r to reduce the number of parameters. Based on DisMult, ComplEx [134] further handles asymmetry thanks to the capabilities of complex embeddings. Instead of tensor product, HolE [98] uses the circular correlation of vectors to represent pairs of entities, i.e., $\mathbf{h}\star\mathbf{t}$ where each element $[\mathbf{h}\star\mathbf{t}]_k=\sum_{i=0}^{d-1}h_i t_{(k+i) \bmod d}$. SimplE [70] is based on canonical Polyadic decomposition [59], which learns an embedding vector for each relation, and a head embedding plus a tail embedding for each

entity. To address the independence between the two embedding vectors of the entities, SimplE introduces the inverse of relations and calculates the average canonical Polyadia score of (h, r, t) and (t, r^{-1}, h).

In addition, ANALOGY [85] supports analogical inference by constraining the relation matrix $\mathbf{M}_r$ to be the normal matrix in linear mapping, i.e. $\mathbf{M}_r^T\mathbf{M}_r = \mathbf{M}_r\mathbf{M}_r^T$. DihEdral [155] employs finite non-Abelian group to account for relation compositions, which supports the non-commutative property of a relation. For example, one exchanges the order between *parent_of* and *spouse_of* will result in different relations (*parent_of* as opposed to *parent_in_law_of*), which indicates the non-commutative property of the relation.

3.1.3 Deep Learning Models

The deep learning models, such as the convolutional neural network (CNN), the recurrent neural network (RNN), and the graph neural network (GNN), are also leveraged as encode functions to embed entities and relations in KGs. For example, ConvE [30] first concatenates a pair of head embedding and relation embedding and then applies 2D convolutions over those embeddings to predict the tail entity. ConvR [68] extends the global filters in ConvE to relation-specific filters, and InteractE [139] captures more interactions by additional convolution operations. Instead of only performing the convolutions on head and relation embeddings, ConvKB [97] applies convolutions over the concatenated embeddings of the head entity, relation, and the tail entity together, which captures the translation relationship in a triplet. CapsE [142] applies the capsule network [118] as the convolution function to encode the entities. RSN [50] integrates RNNs with residual learning to capture the long-term relational dependencies in knowledge graphs.

Recently, GNNs are also attempted to encode the neighboring entities and relations together beyond a single triplet. For example, to adapt to the multiple relations in knowledge graphs, R-GCN extends the transformation weights in GCN to relation-aware weights [120]. To predict the relation between two entities, R-GCN uses relation-aware GCN as the encoder to represent each entity and then leverages DisMult [159] as the decoder to score a given triplet based on the encoded entities and the introduced diagonal matrix $\mathbf{M}_r$ for each relation. Instead of using DisMult as the decoder, SACN [122] uses a variant ConvE [30] as the decoder, where the encoder is also a relation-aware GCN. M-GNN [147] replaces the mean aggregator in each graph convolution layer in R-GCN with a multi-layer perceptron (MLP) to support the injective property, i.e., to map two entities to the same location only if they have identical neighborhood structures with identical embeddings on the corresponding neighbors. In addition to the relation-aware transformation weights, VR-GCN [163] and CompGCN [140] explicitly represent relations and further combine entity and relation embeddings by the operations like subtraction and multiplication. KBGAT [94] enables triplet-aware weights by concatenating the embeddings of the head entity, tail entity, and the relation in a triplet to learn its weight. More details of knowledge graph embeddings can be found in [67], [116].

Summary. The neural reasoning methods utilize shallow embedding models, such as the translation-based models, the multiplicative-based models, or the deep neural network models including CNN, RNN, and GNN, to embed entities and relations in KGs, based on which they perform the reasoning task. These methods are fault-tolerant, as the reasoning is built on the semantic representations rather than the symbolic representations of entities and relations. However, these simple neural network models cannot infer the answers when the complex logic relations $\{\wedge, \vee, \neg\}$ exist between the head and the answer entities. Besides, the neural networks lack interpretation, as they cannot provide explicit rules for explaining the reasoning results.

3.2 Symbolic Reasoning

Symbolic reasoning aims at deducing general logic rules from the knowledge graphs. The entities derived from the given head entity and the query relation following the logic rules are returned as the answers. Existing symbolic reasoning methods are mainly the search-based ILP methods, which usually search and prune rules. This section will first introduce the state-of-the-art search-based method AMIE [42] in detail, and then briefly highlight the improvements of the later methods based on AMIE.

AMIE [42] explores logic rules following two steps. The first step is Rule Extending, which extends candidate rules by three kinds of operations. The second step is Rule Pruning, which prunes the corrupt rules and outputs the confident rules according to the predefined evaluation metrics. For implementation, SPARQL on graph databases is adopted to search the proper facts (h, r, t) in KGs which satisfy the rules extended by the first step and exceed the lower bound of the metrics defined by the second step. The details for each part are as follows:

1) *Rule Extending* is to generate candidate rules by adding three kinds of new atoms into existing rules iteratively, where the first atom is named as the dangling atom which has a fresh variable as one argument and holds an existing variable that appears in other atoms of the rule as the other argument, the second atom is called the instantiated atom which has an argument instantiated by an entity and shares the other argument with other atoms, and the third atom is the closing atom which shares both of the arguments

with other atoms. We present an example of the existing rule and three kinds of atoms as follows:

Rule: $r_h(x,y) \leftarrow r_1(x,z_1) \wedge ... \wedge r_n(z_{n-1},y)$
Dangling atom:$r^D(x,k), r^D(k,y), ...$
Instantiated atom:$r^I(x,K), r^I(K,y), ...$
Closing atom:$r^C(x,z), r^C(z,y), ...$

where $r_i(x,y)$ is an atom that indicates the variable x and y satisfy the predicate r_i. The new atom $r^D(x,k)$ is a dangling atom that shares one variable x with the atoms in the rule and holds a fresh variable k. The new atom $r^I(x,K)$ is an instantiated atom which instantiates one augment with a constant K while sharing the other augment with the rule. The new atom $r^C(x,z)$ shares both x and z with the rule. Due to the fact that the time complexity of extending rules grows exponentially with the increase of the new atoms, a maximal length of the rules is adopted to stop the extending procedure early.

2) *Rules Pruning* applies two metrics, i.e., the head coverage (hc) and the confidence ($conf$), to prune and output the rules. The two metrics are defined as follows:

$$hc(r(x,y) \leftarrow B) := \frac{support(r(x,y) \leftarrow B)}{\#(x',y') : r(x',y')}, \quad (7)$$

$$conf(r(x,y) \leftarrow B) := \frac{support(r(x,y) \leftarrow B)}{\#(x,y) : \exists z_1, ..., z_m : B},$$

where B is the abbreviation of the rule body, i.e., $B = r_1 \wedge \cdots \wedge r_n$, the nominator $support(r(x,y) \leftarrow B)$ indicates the number of the distinct pairs of the head and tail entities (x,y) in KGs (i.e., the facts in KGs) which satisfy the relation r and are derived by the rule body B. The denominator $\#(x^{'},y^{'}) : r(x^{'},y^{'})$ represents the number of the facts in KGs which satisfy the relation r, where the facts might be derived by B or not. The denominator $\#(x,y) : \exists z_1, ..., z_m : B$ with $\{z_1, ..., z_m\}$ as the variables in the rule body B excluding the head and tail variables x and y, denotes the number of the facts that can be derived by the rule body B, where the facts might be observed in KGs or not. Given these notations, $hc(r(x,y) \leftarrow B)$ indicates the proportion of the facts in KGs that are covered by the rule $r(x,y) \leftarrow B$, and $conf(r(x,y) \leftarrow B)$ denotes the proportion of the facts derived by the rule $r(x,y) \leftarrow B$ that are included in KGs. Note the two metrics are based on the close world assumption that the facts not included in the KGs are false. From the definition, we can see that hc represents the coverage/recall of a rule, and $conf$ can reflect the predictive precision of a rule. The two metrics measure the quality of the mined rules in different aspects and are used to prune the rules simultaneously, i.e. the rules whose hc and $conf$ are lower than the predefined thresholds will be discarded and the remaining rules will be outputted.

To implement rule extending and pruning efficiently, the authors project the two processes into a SPARQL query and fire it on KGs:

$$\text{SELECT ?r, WHERE } r_h \wedge r_1 \wedge ... \wedge r_n \wedge \ ?r(X,Y) \quad (8)$$
$$\text{HAVING COUNT}(r_h) \geq K$$

where $?r$ is the new predicate/relation to be added, X and Y represent variables that are either fresh or present in the rule. The above query selects the relation $?r$ such that the result of the query $r_1 \wedge ... \wedge r_n \wedge ?r(X,Y) \rightarrow r_h$ is greater than K. Through setting the proper K, the lower bound of the metrics in Eq.(7) can be satisfied. For example, if choosing K as θ — the lower bound of hc — times the number of the facts in KGs that satisfy the relation r_h, the metric hc of the resulting rules will be greater than θ — which is what we want.

AMIE+ [41] improves the implementation efficiency of AMIE [42] by revising both the Rule Extending process and the metrics defined in the Rule Pruning process.

In the process of Rule Extending, AMIE+ extends a rule only if it is possible to close it[1] before exceeding the predefined maximal rule length. In other words, AMIE+ will not add the dangling atoms at the last step, as this will introduce a fresh variable which results in another non-closed rule. Instead, the instantiated atoms and the closing atoms will be added at the last step to close the rule. The SPARQL queries used to search rules are also simplified. For example, suppose we add a dangling atom to a rule R_p (the parent rule) to produce a child rule R_c, if the predicate/relation of the new atom has already existed in R_p[2], $support(R_c)$ will be the same as $support(R_p)$. Thus it is not necessary to calculate $support(R_c)$ again, which will speed up the SPARQL query process.

In the process of Rule Pruning, calculating either hc or $conf$ requires reckoning the number of the facts derived by a rule. If the rule body contains the atoms with many variables, the derivation process will be expensive. To speed up this, AMIE+ proposes a method to approximate the metrics based on some pre-computed statistics, such as the size of the joins between two relations. Consequently, AMIE+ achieves 100× speedup compared with AMIE.

However, these search-based ILP methods [19], [41], [42] are still not scalable for large KGs, because they

1. A rule is closed if every variable in the rule appears at least twice.

2. AMIE+ allows us to mine the recursive rules, i.e., the head relation occurs in the rule body.

are based on the projection queries implemented by SQL or SPARQL and the huge search space cannot be easily reduced.

RLvLR [102] is one of the early attempts to use the embedding technique to sample the entities and facts that are relevant to the target predicate/relation, which hugely reduces the search space. Specifically, first, RLvLR samples a sub knowledge graph that is relevant to the target predicate; second, it leverages the knowledge graph embedding model RESCAL [100] to generate embeddings for entities and relations in the subgraph, and makes the embedding for an augment of a predicate as the average value of the embeddings of all the entities appearing in the position of this augment; third, it uses a scoring function based on the embeddings to guide and prune the rule search, which turns out to be rather effective for extracting rules; finally, the searched candidate rules are evaluated according to the metrics hc and $conf$ defined in AMIE and are computed by efficient matrix multiplication. By incorporating the embedding technique, the efficiency of the rule searching process is significantly improved.

Statistical measures [19], [41], [42] such as confidence scores may misjudge the rule quality because KGs are inherently incomplete. RuLES [60] leverages the embedding technique to measure the quality of the learned rules. It incorporates the external text information of entities to obtain their embeddings, based on which the confidence score of a fact $r(h,t)$ can be calculated as the dot-product between h and t. Then, RuLES defines the external quality of a learned rule as the average confidence score of all the facts derived by the rule. Finally, both the statistical and embedding measures are intertwined to judge the quality of the learned rules more precisely.

Summary. The traditional search-based ILP methods rely heavily on the search algorithms, various pruning techniques, and efficient database operations, which have several limitations: first, due to the strict matching and the discrete logic operations used during the process of rule searching, the symbolic methods are intolerant of the ambiguous and noisy data; second, the pre-defined evaluation metrics and the rule formations restrict the expressiveness of the learned rules.

3.3 Neural-Symbolic Reasoning

Although symbolic reasoning is good at logical inference and has powerful interpretability, it has difficulties to deal with the uncertainty of entities and relations and the ambiguity of natural languages, i.e., it is not resilient against data noises. On the contrary, the neural networks are fault-tolerant, as they learn the abstract semantics, i.e., embeddings, and further compare these embeddings instead of the literal meaning between entities and relations by symbolic representations. The recent advances on reasoning combine these two kinds of reasoning methods[3]. Typically, there are three main combination methodologies. The first kind targets at neural reasoning but leverages the logic rules to improve the embeddings in neural reasoning, named as *symbolic-driven neural reasoning*. The second kind replaces the neural reasoning with a probabilistic framework, i.e., builds a probabilistic model to infer the answers, where the logic rules are designed as features in the probabilistic model, which is named as *symbolic-driven probabilistic reasoning*. And the third kind aims to infer rules by symbolic reasoning, but incorporates the neural networks to deal with the uncertainty and ambiguity of data. This kind of methods also reduces the search space in symbolic reasoning, which are named as *neural-driven symbolic reasoning*. We will explain the three types of neural symbolic reasoning methods as follows.

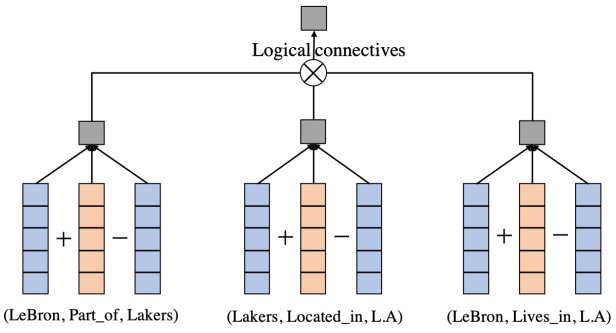


Fig. 2. Illustration of rule scoring in KALE [51].

3.3.1 Symbolic-driven Neural Reasoning

The basic idea of symbolic-driven neural reasoning is to learn the entity and relation embeddings on not only the original observed triplets in KGs but also the triples or ground rules inferred following some pre-defined rules. For example, KALE [51] deals with two types of rules:

$$\begin{aligned} \forall x, y : (x, r_s, y) &\Rightarrow (x, r_t, y), \\ \forall x, y, z : (x, r_{s_1}, y) \wedge (y, r_{s_2}, z) &\Rightarrow (x, r_t, z). \end{aligned} \tag{9}$$

They find all the ground rules of the above two types of rules, assign a score to each ground rule indicating how likely a ground rule is satisfied, and finally learn the entity and relation embeddings on the training set of the original triplets and the ground rules. They employ t-norm fuzzy logics [64], which define the true value of a rule as a composition of the truth values of its constituents through specific t-norm based logical connectives, to calculate a score for a ground rule $f_1 \Rightarrow f_2$ as:

$$s(f_1 \Rightarrow f_2) = s(f_1)s(f_2) - s(f_1) + 1, \tag{10}$$

3. Henceforth, we also name symbolic reasoning as rule learning, and name neural reasoning as embedding learning.

TABLE 2
The format of first-order logic [143]. For example, the third line defines the transitivity rule $(r_1 + r_2) \Rightarrow r_3$, following which we can infer a new triple (e_1, r_3, e_3) from two existing triplets (e_1, r_1, e_2) and (e_2, r_2, e_3).

Triple and ground rule	The format of first-order logic
(h, r, t)	$r(h) \Rightarrow t$
$(h, r_1, t) \Rightarrow (h, r_2, t)$	$[(h \in C) \wedge [r_1(h) \Rightarrow t]] \Rightarrow [r_2(h) \Rightarrow t]$
$(e_1, r_1, e_2) + (e_2, r_2, e_3) \Rightarrow (e_1, r_3, e_3)$	$[[r_1(e_1) \Rightarrow e_2] \wedge [r_2(e_2) \Rightarrow e_3]] \Rightarrow [r_3(e_1) \Rightarrow e_3]$
$(h, r_1, t) \Leftrightarrow (t, r_2, h)$	$[[r_1(h) \Rightarrow t] \Rightarrow [r_2(t) \Rightarrow h]] \wedge [[r_2(t) \Rightarrow h] \Rightarrow [r_1(h) \Rightarrow t]]$

where f denotes an atom, i.e., a triplet, or a formula that is composed of multiple atoms associated by logical operations $\{\wedge, \vee, \neg\}$. If f in the above equation is a triplet, its scored is computed by TransE in Eq.(5). If f is a formula, its score is defined as a composition of the scores of its constituents:

$$\begin{aligned} s(f_1 \wedge f_2) &= s(f_1) \cdot s(f_2), \\ s(f_1 \vee f_2) &= s(f_1) + s(f_2) - s(f_1) \cdot s(f_2), \\ s(\neg f_1) &= 1 - s(f_1). \end{aligned} \tag{11}$$

The procedure of calculating the score of a rule (LeBron, Part_of, Lakers) $\wedge$ (Lakers, Located_in, L.A) $\Rightarrow$ (LeBron, Lives_in, L.A) is illustrated in Figure 2, where the three triplets involved in the rule are scored by TransE, and their scores are combined according to the logical connectives in Eq. (10) and Eq. (11) into the score of the rule. The observed triplets and the rules with high scores are unified as positive instances to learn entity and relation embeddings. The initial rules are selected as the top-ranked ones based on the scores calculated by Eq.(10), where the initial entity and relation embeddings are produced by TransE. The rules will be kept the same during the following embedding learning process.

Based on KALE, Guo et al. further propose RUGE[52] to change the one-round injection of rules into an iterative manner. Instead of directly treating a ground rule as the positive instance by KALE, RUGE injects the triplets derived by some rules as the unlabeled triplets to update the entity/relation embeddings. Since the unlabeled triplets are not necessarily true, the authors predict a probability for each unlabeled triplet based on the current embeddings. Then the embeddings are updated based on both the labeled and unlabeled triplets. The initial rules are obtained by AMIE [42] and also not updated in the following algorithm. In this way, the unlabeled triplet scoring process and the embedding updating process are iteratively computed.

However, KALE and RUGE calculate the score of a rule or a formula as the composition of the scores of its constituents (Cf. Eq.(10) and Eq.(11)), which may result in a high score for a rule or a formula even if the triplets in it are totally irrelevant with each other, as the scores of the triplets are estimated separately. To solve this problem, Wang et al. [143] transform a triplet or a ground rule into first-order logic, and then score this first-order logic by performing some vector/matrix operations based on the embeddings of the entities and relationships included in the first-order logic. Table 2 illustrates the format of the first-order logic, and Table 3 presents how to score the first-order logic by the mathematical expression. In this way, different components, i.e., triplets, included in the same rule have directly interacted in the vector space, which guarantees both the rule and its encoding format have one-to-one mapping transformations.

TABLE 3
Mathematical expression of first-order logic [143].

First-order logic	Mathematical expression
$r(h)$	$\mathbf{r} + \mathbf{h}$
$a \Rightarrow b$	$\mathbf{a} - \mathbf{b}$
$h \in C$	$\mathbf{h} \cdot \mathbf{C}$ ($\mathbf{C}$ is a matrix)
$a \wedge b$	$\mathbf{a} \otimes \mathbf{b}$
$a \Leftrightarrow b$	$(\mathbf{a} - \mathbf{b}) \otimes (\mathbf{a} - \mathbf{b})$

The above methods infer rules one time in the beginning and keep the rules invariant during the learning process. Thus, the rules will impact embedding learning, while the embeddings will not benefit the inference of rules. On the contrary, while IterE [168] also infers new rules based on the updated embeddings at each iteration, it specifically infers new rules and derives new triplets from the rules based on the entity and relation embeddings, and then updates these embeddings based on the extended triplet set. The two processes are executed iteratively. The new confident rules are inferred based on their scores which are calculated by performing some matrix operations over the matrices of the relations included in the rules[4]. To obtain the initial pool of rules, IterE proposes a pruning strategy which has a similar idea as AMIE [42] but combines the operations of traverse and random selection to balance the searching process of potential rules and the convergence of highly possible rules.

3.3.2 Symbolic-driven Probabilistic Reasoning

Symbolic-driven probabilistic reasoning combines the first-order logic and the probabilistic graphical model to learn the weights of logic rules in a probabilistic

4. IterE use DistMult [159] to learn the embeddings, thus a relation is represented as a matrix by DistMult.

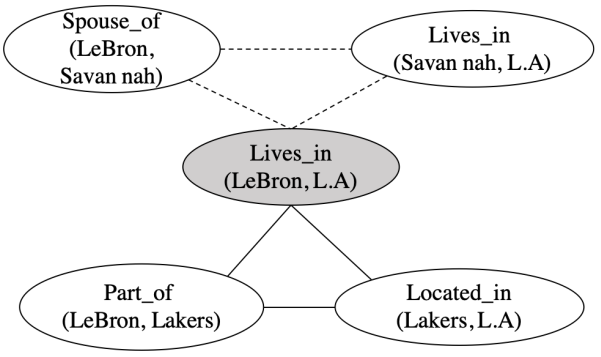


Fig. 3. Two examples of rules and the corresponding ground Markov logic network (Dotted lines are clique potentials associated with rule R_1, and solid lines are with rule R_2. The grey node is the unobserved triplet to be inferred) [146].

framework and thus soundly handles the uncertainty. This kind of methods usually qualify logic rules by firstly grounding the rules, i.e., iteratively substituting variables in the atoms of the rules with any entities in KG until no new facts/triplets can be deduced, and then attaching probabilities to the ground rules. In a sense, the confidence/quality of a logic rule can be defined as the probability distribution over a pre-constructed probabilistic ground graph as shown in Figure 3. Instead of leveraging embedding techniques to qualify the rules, symbolic-driven probabilistic reasoning designs probabilistic models to measure the confidence of rules. In this section, we present two typical probabilistic models, Markov Logic Network (MLN) and ProbLog to explain the characteristics of this category of methods, and then briefly introduce several similar methods.

Markov Logic Network (MLN) [113] is to build a probabilistic graphic model based on the pre-defined rules and the facts in KGs and then learn the weights for different rules. Specifically, given a set of rules $\{\gamma_i\}$, each γ_i can be grounded by the ground atoms (i.e., triplets) from the KGs. Then based on these ground rules, a Markov logic network can be built as follows:

1) A node is built for each ground atom in each ground rule, and the value of the node is set as 1 if the ground atom is observed in KGs, 0 otherwise.
2) An edge is built between two nodes if and only if the corresponding two ground atoms can simultaneously be used to instantiate at least one rule.
3) All the nodes, i.e., ground atoms, in a ground rule form a (not necessarily maximal) clique, which corresponds to a feature, with the value as 1 if the ground rule is true, 0 otherwise. A weight w_i is associated with each rule γ_i.

Take Figure 3 as an example, given two rules R_1 and R_2 together with the observed ground atoms Spouse_of(LeBron, Savan nah), Lives_in(Savan nah, L.A), Part_of(LeBron, Lakers), Located_in(Lakers, L.A) and the unobserved ground atom Lives_in(LeBron,L.A), a Markov logic network can be derived. For example, an edge is built between Lives_in(LeBron,L.A) and Spouse_of(LeBron, Savan nah) as they can simultaneously be used to instantiate R_1. Lives_in(LeBron,L.A), Spouse_of(LeBron, Savan nah) and Lives_in(Savan nah, L.A) form a clique, as they form a ground rule of R_1. With the built Markov logic network, the joint distribution of the values X of all the nodes in the network is defined as:

$$P(X = x) = \frac{1}{Z} \exp(\sum_i w_i n_i(x)), \tag{12}$$

where $n_i(x)$ is the number of true groundings of the rule γ_i and w_i is the weight corresponding to the rule γ_i. Then, MCMC algorithm is applied for inference in MLN and the weights are efficiently learned by optimizing a pseudo-likelihood measure [113]. With the learned weights, we can infer the probability of Lives_in(LeBron,L.A) given its neighboring ground atoms.

However, the inference process in MLNs is difficult and inefficient due to the complicated graph structure among triplets. Moreover, the missing triplets in KGs also impact the inference results by rules. Since the recent embedding techniques can effectively predict the missing triplets and can be efficiently trained with stochastic gradient, pLogicNet [111] proposes to combine the MLN and graph embedding techniques to tackle the above problems. The basic idea is to define the joint distribution of the triplets/facts in KGs with a MLN and associate each logic rule with a weight, but effectively learn them via variational EM algorithm [95]. In the algorithm, E-step infers the plausibility of the unobserved triplets, in which the variational distribution is parameterized by a knowledge graph embedding model such as TransE, while the M-step updates the weights of logic rules by optimizing the pseudo-likelihood defined on both the observed triplets and those inferred by the knowledge graph embedding model.

ProbLog [29] is a probabilistic extension of Programming in Logic[5] (Prolog). Compared with Prolog, ProbLog adds a probability for each clause c_i, which represents either a rule or a ground atom. An example of clauses that are used to derive the query, i.e. finding all the entities e that satisfy Lives_in(LeBron, S), are presented as follows:

5. Prolog is a logic programming language associated with artificial intelligence and computational linguistics. The official website is https://www.swi-prolog.org/.

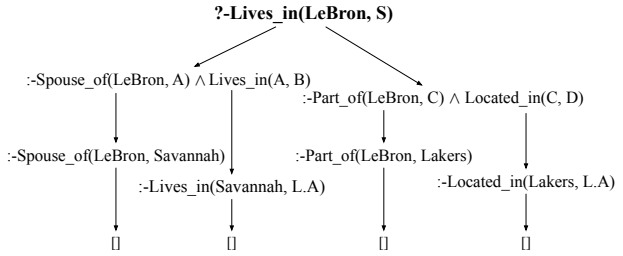


Fig. 4. An example of the SLD-tree built for the query Lives_in(LeBron, S) [24].

$0.7:$ $\text{Spouse\_of}(X, Y) \wedge \text{Lives\_in}(Y, Z) \Rightarrow \text{Lives\_in}(X, Y)$
$0.8:$ $\text{Part\_of}(X, Y) \wedge \text{Located\_in}(Y, Z) \Rightarrow \text{Lives\_in}(X, Y)$
$1.0:$ $\text{Spouse\_of}(\text{LeBron}, \text{Savannah})$
$0.9:$ $\text{Lives\_in}(\text{Savannah}, \text{L.A})$
$0.9:$ $\text{Part\_of}(\text{LeBron}, \text{Lakers})$
$1.0:$ $\text{Located\_in}(\text{Lakers}, \text{L.A})$

Given a query q, the success probability $P(q|T)$ is defined as:

$$P(q|T) = \sum_{L \subseteq L_T} P(q, L|T), \tag{13}$$

$$P(q, L|T) = P(q|L) \cdot P(L|T), \tag{14}$$

$$P(q|L) = \begin{cases} 1 & \exists \theta : L \models q\theta, \\ 0 & otherwise, \end{cases} \tag{15}$$

$$P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L} (1 - p_i), \tag{16}$$

where $T = \{p_1 : c_1, ..., p_n : c_n\}$ depicts a probability distribution over the causes $L \subseteq L_T = \{c_1, ...c_n\}$. Eq.(13) indicates the success probability of query q being decomposed into the summation of all the joint probabilities of the query and each possible cause set L. Eq.(14) further decomposes $P(q, L|T)$ into $P(q|L)$ and $P(L|T)$. $P(q|L)$ represents the probability of the query q given the cause set L, whose value equals 1 if there is at least one answer substitution θ instantiating L and making the query true. Eq.(16) explains how to calculate the probability of a cause set L.

To calculate the success probability $P(q|L)$, a trivial way is to enumerate all the possible logic rules L with their instantiations. Obviously, it is highly inefficient in real-life applications. ProbLog solves this problem by constructing a proving tree for the target query q according to Prolog's Selective Linear Definite (SLD) resolution. The standard SLD-resolution constructs the SLD-tree in a top-down manner, as shown in Figure 4. It first initializes the root node by the query, and then recursively creates subgoals by applying each clause with its instantiations. The iteration stops when reaching the end conditions, for example, the subgoal is empty, which means a possible answer path is found or the maximal tree depth is reached. As a result, each possible answer path is associated with a set of clauses $\{p_1 : c_1, ..., p_n : c_n\} \subseteq T$.

Then following Eq.(13) to Eq.(16), the success probability of a single answer path can be easily computed. A binary decision diagram (BDD) ([13], [37]) is further applied to compute the success probability for multi-paths. SLD-tree is also the base of other similar methods such as stochastic logic programs (SLPs) [24] and Programming with Personalized PageRank (ProPPR) [146].

Both MLN and ProbLog learn the probabilities for rules, where MLN builds a global probabilistic graph for all the rules and learns the probabilities for all the rules simultaneously, but ProbLog constructs a local SLD-tree for each query and learns the probabilities for the causes that can support the target query. Other similar methods, such as probabilistic Datalog [40], MarkoViews [65], stochastic logic programs (SLPs) [24], also attach probabilities to clauses, but having variant optimization frameworks when updating these probabilities. For example, SLPs define a randomized procedure for traversing the SLD-tree, where the probability distribution defined over nodes is learned by up weighting the desired answer clauses and down weighting the others. Programming with Personalized PageRank (ProPPR) [146] is an extension to SLPs which changes the randomized sampling to a bias-based strategy based on personalized PageRank (PPR) [15], [103]. They use PPR to calculate the probability for each clause based on some pre-defined features, instead of directly setting a probability in Eq.(16).

3.3.3 Neural-driven Symbolic Reasoning

Neural-driven symbolic reasoning aims to derive the logic rules, where the neural networks are incorporated to deal with the uncertainty and the ambiguity of data, and also reduce the search space in symbolic reasoning. The basic idea is to extend the multi-hop neighbors around the head entity and then predict the answers included in these neighbors. We further divide neural-driven symbolic reasoning methods into path-, graph- and matrix-based framework according to the number of the extended neighbors in each step.

Path-based Reasoning. Path-based reasoning extends only one neighbor at each step. Path-Ranking Algorithm (PRA) [78] is a state-of-the-art path-based method. Given a head entity h and a tail entity t, PRA obtains the paths of length l from h to t by performing a random walk with restart algorithm of l steps and then calculates the score $s_p(h, t)$ of the entity pair (h, t) following the path p. Finally, PRA estimates the weights of different paths by a linear regression model with $s_p(h, t)$ of different paths as the corresponding feature values. Essentially, PRA is pure symbolic reasoning. We place PRA in this section

because most of the following neural-driven symbolic reasonings are based on the idea proposed by PRA.

PRA is highly dependent on the connectivity of KGs because PRA cannot predict the relation between the nodes that are disconnected from each other. To deal with the problem, Lao et al. [79] performs PRA on the graph which consists of both the edge from the original KGs and the syntactic dependency edges extracted from the dependency-parsed web text. Instead of using the above-unlexicalized dependencies, Gardner et al. [47] use the lexicalized words extracted from the corpus to build additional edges, which is more expressive. However, all the above frameworks need to train a model for each relation type and cannot be generalized to other unobserved relations in the test set.

Based on the set of paths connecting the head and tail entities found by PRA, Neelakantan et al. [96] aims to find the most confident path for the entity pair, which is the one with the highest similarity between the embedding of the path and the embedding of the target relation. The embedding of a path is computed by applying the composition function based on the embeddings of the relations included in the path recursively following RNN. Finally, the predictive score of a fact/triplet is represented as the dot product between the embedding of the most confident path and the embedding of the relation. The incorporation of RNN improves the generalization of the model, which can be used to deal with the new relation types that are unobserved in the training data. Instead of only using the most confident path to predict the score of a triplet, Chain-of-Reasoning [27] combines the similarities of different paths by different strategies such as Top-k, Average, and LogSumExp.

However, the paths are traversed heuristically, which lacks evaluation in the above methods. Meanwhile, when the number of relations in KGs is large, even with the constraint that paths are at most three steps, the number of finding paths will easily reach millions, so it is impractical to enumerate all possible relation paths. Thus, DeepPath [153] proposes a reinforcement learning (RL) method to evaluate the sampled paths, which can reduce the search space. The basic idea is to frame the multi-hop reasoning as a Markov Decision Process (MDP) and design a policy function to encode the continuous state of the RL agent. The agent reasons in the vector space environment by sampling a relation at each hop to extend the reasoning path. The state is composed of the embeddings of the current entity and the target entity. The reward function is designed to measure the accuracy, efficiency, and diversity of the path, which will supervise the sampling action at each hop. Meanwhile, to tackle the problem of large action space, DeepPath is initialized by supervised training, where paths traversed by a two-way BFS algorithm are to guide the RL agent. AnyBRUL [90] also uses RL to sample paths. Additionally, after obtaining the sampled paths, it constructs the ground rules from the sampled paths and generalizes them to abstract rules according to a bottom-up approach.

DeepPath [153] and AnyBRUL [90] require to first sample all the paths between the head and the tail entities, and then leverage them to evaluate whether the tail entity can be the right answer, so, it cannot be directly applied to the scenario where the tail entities are unobserved. Thus, instead of adopting RL to infer paths, some methods choose to train RL directly to obtain the correct answer entity by the given head entity and the query relation. Among these models, MINERVA [26] is a representative model that samples each neighbor according to a LSTM-based policy function. Different from DeepPath, the state in MINERVA is composed of the embedding of the query relation and the partial path, and the embedding of the answer entity is not needed during the sampling process. These allow MINERVA to derive the answers directly. MINERVA also incorporates a self-relation for each entity to indicate the stop action when the entity itself is sampled. A hard reward $0/1$ indicating if the sampled entity at the last step is the correct answer is used to supervise the sampling process.

Instead of using the hard $0/1$ reward, Multi-Hop [83] proposes a soft reward based on the similarity between the true answer entity and the sampled entity at the last step. In addition, inspired by the dropout technique, the model masks some actions by chance during training to avoid choosing lots of repeated paths and alleviates the problem of overfitting. CPL [39] leverages the text information in addition to the graph structure of the KGs during the sampling process. Specifically, when determining the next-hop entity, it not only considers the entities in KGs, but also leverages the entities extracted from the text corpus. M-walk [124] introduces a value-based RL method and uses Monte Carlo Tree Search (MCTS) to overcome the challenge of sparse positive rewards. Specifically, it adopts an MCTS trajectory-generation step and a policy-improvement step iteratively to refine the policy function, which achieves more positive rewards. DIVA [17] frames the KGC task into a unified model which consists of the components of path finding and answer reasoning, where the paths are modeled as hidden variables and VAE [71] is adopted to solve the model. Compared with DIVA, PRA, and Chain-of-Reasoning, DeepPath and AnyBRUL emphasize the process of path finding to explicitly derive rules and leave the answer reasoning process to an additional step, while MINERVA, Multi-hop, and M-walk directly reason the answers without explicitly deriving the rules.

Figure 5 summarizes the idea of the path-based reasoning process, where $q = (h, r)$ indicates the head entity h and the query relation r, n_t represents the current entity, a_{t-1} is the last action that results in

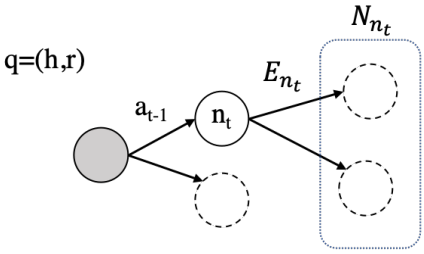


Fig. 5. Illustration of the RL-based reasoning [124].

the current entity n_t, E_{n_t} and N_{n_t} are the set of the relations and the corresponding neighbors of n_t, from which the next relation and entity are selected.

Graph-based Reasoning. Graph-based reasoning methods are the extensions to the path-based reasoning methods, which could better explain reasoning in KGs by structuring explanations as a graph rather than a path.

GraIL [130] is a graph neural network performed on an extracted subgraph to reason the relation between two entities. Specifically, to validate a triplet (h, r, t), GraIL first extracts the subgraph around h and t as the intersection of the k-hop neighbors of the two entities and the relations among them. Then it labels the entities in the extracted subgraph with tuple $(d(i, h), d(i, t))$, where i denotes the i-th node in the subgraph and $d(u, v)$ denotes the shortest distance between u and v. Next, it adopts an attention-based multi-relational GNN model, i.e., R-GCN [120], to compute the embedding for each node in the subgraph. And finally, the model concatenates the embeddings of the subgraph (the average of the embeddings of all the entities in the subgraph), the head entity, the tail entity, and the query relation, based on which it scores the correctness of the given triplet (h, r, t). GraIL reasons over local subgraph structures and has a strong inductive bias to learn entity-independent relational semantics.

Starting from the given head entity and the query relation represented by a pair of head and tail entity, CogGraph [34] extends multiple entities at each step by a policy function, and then proposes a GNN model to embed each node in the extended subgraph. CogGraph predicts the answers based on their embeddings. DPMPN [152] proposes two GNN models to perform the reasoning. The first GNN model performs the input-invariant message passing globally on the whole KG, which can provide raw and rich representations for entities. The second GNN model performs pruned message passing locally on a subgraph related to the query, which captures input-dependent semantics, disentangled from the full graph.

Matrix-based Reasoning. Matrix-based reasoning can be viewed as an extension to graph-based reasoning, which does not select the neighbors at each hop but incorporates the soft attention scores to indicate the importance of different neighbors. The basic idea is to express the logical relationships between the head and the tail entities by matrix operations. This section firstly introduces the earliest attempt — TensorLog [22], and then describes three typical models, Neural LP [160], Neural Logic Inductive Learning (NLIL) [162] and Neural-Num-LP [144]. Based on Tensorlog, Neural LP further learns the new rules, NLIL has the capacity of expressing complex rules shown in Figure 6, and Neural-Num-LP [144] particularly deals with the numerical operations such as comparison, aggregation, and negation among entities.

TensorLog [22] infers the weighted chain-like logical rules for explaining each relation R in KGs. Then based on the inferred rules, given a query $R(x, Y)$ with the query relation R and the head entity x, the goal is to retrieve a ranked list of entities such that the ground truth answer y is ranked as high as possible.

In TensorLog, each entity in KGs is represented as a one-hot embedding, and each relation R in KGs is represented as a matrix M_R, where each element $M_R[i, j] = 1$ if the fact $R(i, j)$ is in the KGs. Then given a rule γ such as $R(X, Y) \rightarrow P(X, Z) \wedge Q(Z, Y)$ and a head entity x, the logical inference of the answers can be implemented by performing matrix multiplications $M_P \cdot M_Q \cdot v_x$. The result is a vector with each nonzero element y indicating that there exists z such that $P(y, z)$ and $Q(z, x)$ are in the KGs. Since a query relation may be explained by multiple rules, the score of the query relation is calculated by combining all the rules:

$$\sum_{\gamma} \alpha_{\gamma} \prod_{k \in \beta_{\gamma}} \mathbf{M}_{R_k}, \tag{17}$$

where γ indexes over all possible rules, α_{γ} is the confidence associated with the rule γ, and β_{γ} is an ordered list of all predicates in the rule γ. During inference, given a head entity x, the score of each retrieved answer equals to the entries in the vector $\mathbf{s}$:

$$\mathbf{s} \quad = \quad \sum_{\gamma} (\alpha_{\gamma} (\prod_{k \in \beta_{\gamma}} \mathbf{M}_{R_k} \mathbf{v}_x)). \tag{18}$$

Then the probability of each true answer y satisfying $R(x, y)$ given the head entity x is maximized:

$$\max_{\{\alpha_{\gamma}, \beta_{\gamma}\}} \sum_{x,y} \text{score}(y|x) = \tag{19}$$

$$\max_{\{\alpha_{\gamma}, \beta_{\gamma}\}} \sum_{x,y} \mathbf{v}_y^T \left(\sum_{\gamma} \alpha_{\gamma} (\prod_{k \in \beta_{\gamma}} \mathbf{M}_{R_k} \mathbf{v}_x) \right)$$

where $\mathbf{v}_x$ and $\mathbf{v}_y$ indicate the one-hot vectors of the head entity x and the ground truth answer y, respectively.

To learn the parameters α and β, TensorLog formalizes each rule into a factor graph, where each node in the graph represents a variable in the rule and each edge indicates a predicate/relation in the rule. Given a factor graph, or a rule that explains the query $R(x, Y)$, the probability of an answer y to the query, i.e., $R(x, y)$, can be defined as the joint probability of all the possible grounding entities in the graph with x and y being fixed. The approximation algorithm belief propagation [49] is adopted to compute the probability.

TensorLog estimates the rule confidence in a differentiable matrix-production manner. However, it is unable to generate new logic rules.

Neural Logic Programming (Neural LP) [160] improves the matrix-based reasoning framework based on TensorLog. In TensorLog, learning parameters is difficult because each rule is associated with a parameter, and enumerating rules is an inherently discrete task. To overcome this difficulty, Neural LP interchanges the summation and the product in Eq.(17) and decomposes the weight of a rule into the weights of the predicates in the rule. The concrete formula to score a query relation is:

$$\prod_{t=1}^{T} \sum_{k}^{|R|} a_t^k \mathbf{M}_{R_k}, \tag{20}$$

where T is the maximal length of the rule and $|R|$ is the number of the predicates in the KGs. However, the expressive capability of Eq.(20) is not sufficient enough, as it assumes that all the rules are of the same length T. To address this issue, Neural LP designs a recurrent formulation to model the length of rules dynamically. It is defined as follows:

$$\begin{aligned} \mathbf{u}_0 &= \mathbf{v}_x \\ \mathbf{u}_t &= \sum_{k}^{|R|} a_t^k \mathbf{M}_{R_k} \Big(\sum_{\tau=0}^{t-1} b_t^\tau \mathbf{u}_\tau\Big) \quad \text{for } 1 \le t \le T \\ \mathbf{u}_{T+1} &= \sum_{\tau=0}^{T} b_{T+1}^\tau \mathbf{u}_\tau \end{aligned} \tag{21}$$

where $\mathbf{u}$ is an auxiliary memory vector initialized as the input entity $\mathbf{v}_x$. At each step, the model first computes a weighted average of previous memory vectors using the memory attention vector b_t, and then "softly" applies the TensorLog operations using operation attention vector a_t. Finally, it computes a weighted average of all the memory vectors and uses attention to control the length of rules.

TensorLog and Neural LP aim at learning some new probabilistic chain-like logic rules for knowledge bases reasoning, but they fail to infer some complex formations of rules such as tree-like, conjunctions, etc. (Refer to the examples in Figure 6). Moreover, the rules are inferred based on the specific head entity x, which deteriorates the generalization of the learned rules.

Neural Logic Inductive Learning (NLIL) [162] tackles the non-chain-like rules by incorporating a primitive statement. Specifically, the concept of a primitive statement is an extension to the concept of atom. An atom is defined as a predicate applied to the logic variables, while a primitive statement is defined as a predicate applied to the logic variables or the results of some operators. Here, an operator is defined upon a predicate:

$$\begin{cases} \varphi_k() = \mathbf{M}_k \mathbf{1} & \text{if } k \in \mathcal{U}, \\ \varphi_k(\mathbf{v}_x) = \mathbf{M}_k \mathbf{v}_x & \text{if } k \in \mathcal{B}, \end{cases} \tag{22}$$

where $\mathcal{U}$ and $\mathcal{B}$ are the sets of unary and binary predicates respectively. The operator of the unary predicates takes no input and is parameterized with a diagonal matrix. For example, for the unary predicate such as Person(X), its operator $\varphi_{Person}() = \mathbf{M}_{Person}\mathbf{1}$ returns the set of all entities labelled as person. The operator of the binary predicates returns the tail entities that, together with the head entity, i.e., the input of the operator, satisfy the predicate P_k. For example, for the binary predicate such as Mother(X,Y), its operator $\varphi_{Mother}(\mathbf{v}_x)$ returns the mothers of the input variable x.

Based on the definition of the operator, the primitive statement can be instantiated as follows:

$$\psi_k(x, y) = \begin{cases} \sigma((\mathbf{M}_k \mathbf{1})^{\mathrm{T}} (\prod_{t'=1}^{T'} \mathbf{M}^{(t')} \mathbf{v}_y)) & \text{if } k \in \mathcal{U}, \\ \sigma((\prod_{t=1}^{T} \mathbf{M}^{(t)} \mathbf{v}_x)^{\mathrm{T}} (\prod_{t'=1}^{T'} \mathbf{M}^{(t')} \mathbf{v}_y)) & \text{if } k \in \mathcal{B}, \end{cases} \tag{23}$$

where σ is the sigmoid function. In Eq.(23), unary primitive statements and binary primitive statements are instantiated in different ways. Unary primitive statements contain only one relation path starting from variable y, while binary primitive statements contain two relation paths starting from variable x and y respectively. Compared with the multiplication of the predicates in a single relation path in Eq.(19), Eq.(23) replaces the answer vector $\mathbf{v}_y$ with another relation path, which makes it possible to represent "correlations" between two variables, and the path that starts from the unary operator, e.g., φ_{Person}. In this way, a primitive statement is capable of representing the tree-like logical rules, as shown in Figure 6. Then similar to Eq.(21), Eq.(23) can also be relaxed into weighted sums. Instead of assigning a single path attention vector for all the relation paths in Eq.(19), NLIL assigns separate attention vectors for each relation path in the k-th primitive statement.

Then the logic combinations of primitive statements, via $\{\wedge, \vee, \neg\}$, as shown in Figure 6, are represented by the following equations:

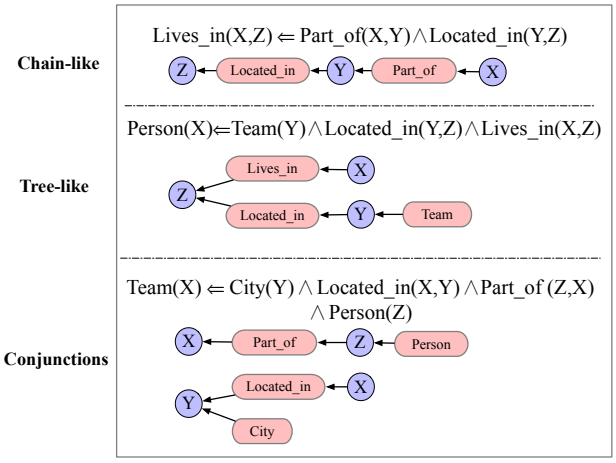


Fig. 6. Examples of chain-like, tree-like and conjunctions of rules. Refer to [22].

$$\mathcal{F}_0 = \Phi, \tag{24}$$

$$\hat{\mathcal{F}}_{l-1} = \mathcal{F}_{l-1} \cup \{1 - f(x,y) : f \in \mathcal{F}_{l-1}\}, \tag{25}$$

$$\mathcal{F} = \{f_i(x,y) * f_i'(x,y) : f_i, f_i' \in \hat{\mathcal{F}}_{l-1}\}_{i=1}^{2}, \tag{26}$$

where each element in the formula set $\{f : f \in \mathcal{F}_l\}$ is called a formula which accepts two entities as input and outputs a scalar. The initial formula set contains only primitive statements Ψ. The $(l-1)$-th formula set $\mathcal{F}_{l-1}$ is concatenated with its logic negation to yield $\hat{\mathcal{F}}_{l-1}$. Then each formula in the next level is the logic $\wedge$ of two formulas from $\hat{\mathcal{F}}_{l-1}$, as the logic $\vee$ can be implicitly represented as $p \vee q = \neg(\neg p \wedge \neg q)$. Similarly, the formula selection can be parameterized into the weighted-sum form with the attentions.

In practice, the above different attention vectors are learned by three stacked transformer networks. This complex and compact framework has the capacity of discovering more expressive underlying reasoning patterns.

All reasoning models discussed above only focus on the relational structures of KGs but ignore the numerical values of entities that may be involved in the reasoned rules.

Neural-Num-LP [144] extends Neural LP [160] to learn the numerical rules. It supports the comparison operator by defining it in a matrix format:

$$(M_{r_{\overline{pq}}^{\leq}})_{ij} = \begin{cases} 1 & \text{if } p_i \leq q_j, \\ 0 & \text{otherwise,} \end{cases} \tag{27}$$

where p, q are two numerical features such as "hasCitation" and "birthDate". This matrix denotes the binary indicator of the comparison over all pairs of entities in KGs containing the features p and q, which can be multiplied with other predicate/relation matrices presented in Eq.(17).

Neural-Num-LP is a good inspiration for fully understanding the reasoning patterns not only from the relations but also from the attributes.

Summary. The first kind of neural-symbolic reasoning, i.e., the symbolic-driven neural reasoning, aims to learn the entity and relation embeddings. The logic rules are used to increase the number of highly confident triplets to improve embedding performance. Thus the reasoning process is still based on embeddings, which lacks interpretation.

The second kind, i.e., the symbolic-driven probabilistic reasoning, qualifies the logic rules by grounding the rules in KGs. With the increase of the entities and relations in KGs, the grounding atoms/rules will increase dramatically, making inference and learning computationally expensive. Besides, these methods cannot produce new rules.

The above two kinds of methods take the answer prediction as the only target. The difference is that in symbolic-driven probabilistic reasoning, the rules are used as the features to predict the answers, while in symbolic-driven neural reasoning, the rules are used to generate more facts for learning high-quality embeddings.

The third kind, i.e., the neural-driven symbolic reasoning, takes both the answer prediction and the rule learning as the target. To achieve the goal, it infers the answers following the paths, graphs, or matrixes started from the head entity, which can enhance the interpretability of the predicted answers. However, with the increase of the hop number, the paths, subgraphs, or the matrix multiplication become more complex, making the predictive performance more sensitive to the sparsity of the knowledge graphs.

4 Reasoning for KGQA

In this section, we will introduce the reasoning methods for KGQA, which can also be categorized into the neural, the symbolic and the neural-symbolic reasoning methods. KGQA needs to deal with the natural language questions, which is more complex than reasoning for KGC. Thus according to the types of the questions, we can further categorize KGQA into simple-relation questions, multi-hop relation questions and complex-logic questions.

Single-relation questions refer to questions that only involve a single topic entity and a single relation in KGs. Then the tail entities in KGs corresponding to the topic entity and the relation are extracted as the answers. Example questions of this type include: "Who is the wife of Barack Obama" or "Where is the Forbidden City". Multi-hop relation questions are path-based which means the answer can be found by walking along a path consisting of multiple intermediate relations and entities starting from the topic entity. Complex-logic questions contain several subject entities aggregated by conjunction ($\cap$), disjunction ($\cup$)

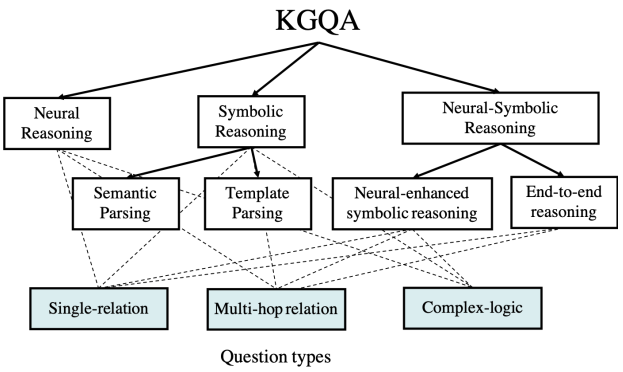


Fig. 7. The taxonomy of the KGQA reasoning methods. The dashlines link the question types to the methods that can address them.

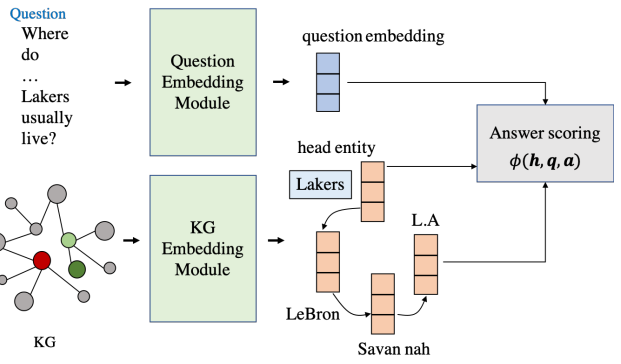


Fig. 8. Illustration of the neural reasoning method EmbedKGQA for solving the multi-hop relation questions [119].

or logical negation ($\neg$), which means the answer can be obtained by some operations, such as intersection of the results from multiple path queries (multi-hop questions). Questions may also have some complex constraints (Cf. Table 5 for details). Figure 7 presents the taxonomy of the KGQA reasoning methods and the links between the question types and the methods that can address them.

4.1 Neural Reasoning

Neural reasoning methods for question answering usually encode the entities and relations in KGs as well as the input questions into embeddings of the same space, based on which they infer the answers. Neural reasoning methods can deal with three kinds of questions: the single-relation questions, the multi-hop relation questions and the complex-logic questions.

Single-relation Question. KEQA [63] deals with the single-relation questions by finding several candidate triplets. The tail entity in the candidate triplets is chosen as the answer according to the embeddings of the triplet, the topic entity and the query relation in the question. Specifically, to find the candidate triplets, KEQA first trains a head entity learning model with Bidirectional LSTM as the key component to predict the entity token and the non-entity token in the question literals. It then extracts the topic entity based on the predicted tokens and searches the entities in KGs which are same to the topic entity or contain the topic entity as the candidate heads. All the triplets with head entity in the candidate heads are named as the candidate triplets. Next, to obtain the embeddings of the topic entity and the query relation, the authors propose another bi-directional LSTM which takes the sequential tokens in the question literals as input to predict the corresponding embedding. Finally, for each candidate triplet, KEQA minimizes the Euclidean distance of the predicted embedding between the candidate and the ground truth, in head entity, predicate, and answer.

Liu et al. [88] propose a similar end-to-end method to address the single-relation questions. For topic entity and single relation, both word-level information and character-level information from the question are considered. Other researches [25], [48], [93], [136], [165] are the similar neural reasoning methods for single-relation KGQA.

Multi-hop Relation Question. In addition to single-relation questions, EmbedKGQA [119] is proposed to deal with the multi-hop relation questions. It employs ComplEx [135] to embed entities and relations in the complex space and applies the same ComplEx scoring function to predict the answers. Specifically, the input question q is first embedded by RoBERTa [87], and then projected by a feed-forward neural network into the complex space. Then a score $\phi(\mathbf{h}, \mathbf{q}, \mathbf{a})$ is calculated for each triplet of a topic entity, a question and an answer such that $\phi(\mathbf{h}, \mathbf{q}, \mathbf{a}) > 0$ if a is an answer entity and $\phi(\mathbf{h}, \mathbf{q}, \mathbf{a}) < 0$ for a is not, where $\mathbf{h}$, $\mathbf{q}$ and $\mathbf{a}$ are the embeddings for the topic entity, question and answer respectively. EmbedKGQA selects the entity with the highest score as the answer. Figure 8 illustrates the basic idea of EmbedKGQA. Other works such as [20], [32], [101] are similar works for multi-hop KGQA.

Complex-logic Question. KEQA and EmbedKGQA cannot handle complex logical questions, because these queries involve the logical operations that will result in multiple entities at each hop. To address this defect, some researches represent the logic operations as the learned geometric operations. For example, Hamilton et al. [53] propose a neural reasoning model GQE (i.e., graph query embeddings) to deal with the conjunctive logic queries. On the basis of their work, Q2B (QUERY2BOX) proposed by Ren et al. [112] fills the gap in disjunctive queries. Both GQE and Q2B assume that a complex logical question can be represented as a DAG. They both start with the embeddings of the topic entities and then iteratively

TABLE 4
A summary of knowledge graph question answering and recent advances. S: single-relation question, M: multi-hop relation question, C: complex-logic question.

Category	Sub-category	Question Type	Model	Mechanism
Neural Reasoning	-	S	KEQA [63]	train a topic entity learning model and query relation learning model
		S	Liu et al. [88]	train an end-to-end topic entity and query relation learning model
		M	EmbedKGQA[119]	use RoBERTa to embed the question
		C	GQE [53]	deal with conjunctive logic queries
		C	QUERY2BOX [112]	deal with disjunctive queries
		C	EmQL [126]	obtain faithful embeddings
Symbolic Reasoning	Semantic Parsing	C	Kwiatkowksi et al. [75]	follow CCG to parse questions
		C	Berant et al. [5]	follow λ-DCS to parse questions
		C	Dubey et al. [35]	follow NLQF to parse questions
	Template-based Parsing	C	UncertainTQA [171]	link query graphs and SPARQLs as templates
		C	QUINT [1]	link query graphs and dependency parse trees as templates
		C	TemplateQA [170]	link natural language patterns and SPARQLs as templates
Neural Symbolic Reasoning	Neural-Symbolic Reasoning	S	Yih et al. [164]	link questions to KG and use CNN to encode the linked entities/relations
		M	MULTIQUE [7]	generate a sub query graph and use LSTM to qualify it
		C	Bao et al. [2]	generate a multi-constraint query graph
		C	Lan et al. [77]	incorporate constraints and extend relation paths simultaneously
	End-to-end Reasoning	M	IRN [172]	extend a path based on question and entity/relation embeddings
		M	SRN [110]	use RL to learn the path sampling strategy
		M	Graft-Net [127]	extract a subgraph and apply GNN to infer the answer in it
		M	PullNet [128]	use RL to learn the subgraph sampling strategy
		M	VRN [169]	model the topic entities as hidden variables

apply their proposed geometric operations to generate the embedding of the query, which are then used to predict the answer entities according to their similarities.

The two proposed geometric operators in GQE are the geometric projection operator P and the geometric intersection operator I, where P is responsible for projecting a query embedding $\mathbf{q}$ of the last hop according to the relations of the outgoing neighbors to obtain the embedding $\mathbf{q}'$ of the current hop, and I aggregates the embeddings of all the incoming neighbors of a node in the DAG to simulate the logic conjunction operator. Specifically, P and I are implemented as:

$$\begin{aligned} P(\mathbf{q}, r) &= \mathbf{R_r}\mathbf{q}, \\ I(\{\mathbf{q}_1, ..., \mathbf{q}_n\}) &= \mathbf{W_r}\Psi(NN_k(\mathbf{q}_i), \forall i = \{1, ..., n\}) \end{aligned} \tag{28}$$

Where $\mathbf{R_r}, \mathbf{W_r}$ are trainable parameter matrices for relation r, NN_k is a k-layer feedforward neural network and Ψ is a symmetric vector function (e.g., an elementwise mean or min of a set over vectors).

However, GQE embeds a question into a single point in the vector space, which is problematic because there are usually multiple immediate entities when traversing the KGs for answering questions. And it is not clear how to effectively model a set with a single point. Also, it is unnatural to define the logical operation of two points in the vector space. Thus, Q2B is proposed to embed a query as a box in which the set of points correspond to the answer entities of the query. A box is represented by the embedding of the center and the offset of the box, which models a set of entities whose vectors are inside the box. Each relation is associated with a box embedding. Based on the definition of the box, given an input box embedding $\mathbf{p}$ and a relation embedding $\mathbf{r}$, the geometric projection operator can be simply represented as $\mathbf{p} + \mathbf{r}$, where the centers and the offsets of them are summed respectively. Then the intersection of a set of box embeddings is calculated by performing attention over the box centers and shrinking the box offset using the sigmoid function. As for the union operation, since boxes can be located anywhere in the vector space, the union of two boxes would no longer be a simple box. To rectify this issue, Q2B transforms the query into Disjunctive Normal Form (DNF), i.e. disjunction of conjunctive queries, such that the union operation only appears in the last step. Then a DNF query can be solved by firstly representing each conjunctive query and then simply aggregating their results, for example, taking the nearest point of the boxes of all the conjunctive queries. However, the negation operation is not involved in the above two models.

Both GQE and Q2B learn geometric operators to simulate logic operators, which, however, are not faithful to deductive reasoning and fail to find entities logically entailed as answers, due to the fact that they casually use spatial operations hoping to achieve logic reasoning. Thus, EmQL [126] can obtain faithful embeddings by proposing five operators for reasoning, including set intersection, union, difference, relation following and relational filtering.

Summary. Although the neural reasoning methods can deal with the three kinds of questions, the complex questions with different constraints are not totally solved by the neural reasoning methods (Cf. Table 5 for details). And similar to KG completion, the

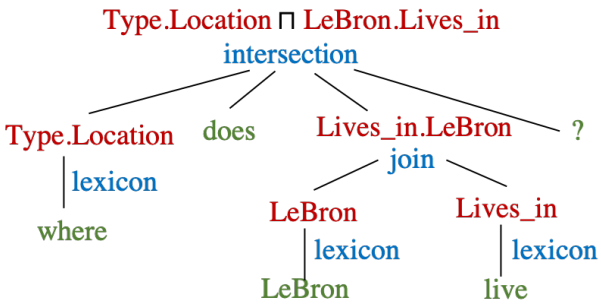


Fig. 9. An example of transforming the question "Where does LeBron live?" into a logic expression following λ-DCS [5].

neural reasoning methods also lack the interpretation for KGQA.

4.2 Symbolic Reasoning

Symbolic reasoning methods for KGQA fall into two major categories — semantic parsing ones and template-based ones. Both aim at generating structured queries for unstructured natural language questions. However, they differ in the way to understand the natural language questions. The semantic parsing methods employ NLP tools to convert the question into the syntactic dependency representation, while the template-based methods use a large number of templates which consist of both the natural language pattern and the corresponding structured query pattern like SPARQL to decompose the complex question.

Semantic Parsing. Semantic parsing methods target at parsing the input natural language question into a logic expression, based on which the answer to the question can be easily retrieved from the KGs. Different semantic parsing methods transform the natural language questions into different forms of logic expressions.

For example, Kwiatkowksi et al. [75] follow combinatory categorial grammar (CCG) [11], a linguistic formalism that couples syntax and semantics, to transform the questions. For example, for a sentence x "New York borders Vermont", following the CCG grammar, its corresponding logic expression is Next_to(ny, vt). The target is to learn a function that maps a sentence x to the logic expression z, where the function is learned by inducing CCG from the training data of $\{(x, z)\}$. The whole algorithm consists of two components, where the first one learns the CCG lexicon that is used to define the space of the predicates in the logic expressions, and the second one learns the parameters of the features reflecting the probability of the logic expressions.

Berant et al. [5] follow Lambda Dependency-based Compositional Semantics (λ-DCS) [82] to transform the questions, in order to make existential quantification implicit, thus reducing the number of variables. For example, $\lambda x.\exists a.p_1(x, a) \wedge \exists b.p_2(a, b) \wedge p_3(b, e)$ is expressed compactly as $p_1.p_2.p_3.e$ by hiding the existential quantifications λ, a and b. We give a real example of transforming the question "Where does LeBron live" into the logic expression following λ-DCS in Figure 9, where the blue labels indicate the composition rules and the red nodes denote the transformed logic expressions. The whole algorithm also consists of two components, where the first one maps the natural language phrases to the predicates in KGs based on some pre-defined lexicon mappings and a set of composition rules, and the second one is a log-linear model to learn the parameters of the features which reflect the probability of the transformed logic expressions. This method differs from the method presented in [75] in two ways. First, they propose a bridging function to deal with the ambiguity of the predicates in natural language questions through generating predicates compatible with the neighboring predicates. Second, the logic expressions $D(x) = \{d\}$ are assumed to be unobserved, and only the question-answer pairs $\{(x_i, y_i)\}$ can be obtained to supervise the logic transformation. Thus, the log-likelihood of the correct answer $(d.z = y_i)$, summing over the latent logic expression d, is optimized:

$$\mathcal{O}(\theta) = \sum_{i=1}^{n} \log \sum_{d \in D(x): d.z = y_i} p_\theta(d|x_i), \tag{29}$$

where $p_\theta(d|x_i)$ is the probability of generating the logic expression d from the sentence x_i given the parameters θ.

Dubey et al. [35] follow natural language query formalization (NLQF) to transform the questions. They introduce a chunker-styled pseudo-grammar, named Normalized Query Structure (NQS), to parse each token in the natural language questions. Instead of using static templates, its syntax definition is dynamically fitted to the natural language sentence. For example, the question "Desserts from which country contain fish." is finally parsed as "[wh = which][R1 = null][D = country][R2 = from][I1 = dessert][R3 = contain][I2 = fish]". Once the NQS instance is obtained, it is fed into the NQS2SPARQL module to generate the SPARQL query, where the SPQRAL is a semantic query language for databases. This module analyzes the NQS instances and then maps the entities in the NQS instances to the entities in KGs.

Other similar semantic parsing methods can be referred to [69], [114], [121], [149], [156], [157], [167], [173].

Template-based Parsing. Template-based parsing usually contains an offline stage and an online stage, where the offline stage targets at generating a collection of templates and the online stage aligns the new arriving question to an existing template, and then aims to retrieve the answer from the KGs based on the matched template. The templates can be mainly

summarized into two forms: one in natural language pattern and the other in the structured graph pattern. Both of them correspond to the structured queries such as SPARQL, which can be directly executed on KGs to retrieve the answers. Thus there are two main problems in template-based parsing methods: the first one is how to generate the templates and the second one is how to match questions with templates.

As one of the pioneering work, Bast and Haussmann [3] manually construct three simple question templates without QA corpora. Since it is expensive to manually define these templates, especially for open-domain QA systems over large KGs, later work studies how to generate templates automatically. For example, Zheng et al. propose UncertainTQA [171] to generate templates from the collected natural language questions and SPARQL queries on KGs. It first employs the existing method such as [174] to translate each natural language question into a semantic query graph g. Meanwhile, a SPARQL query graph q can be also constructed for each SPARQL query. Then graph edit distance (GED) is used to calculate the graph similarity between the query graph g and the SPARQL query graph q. A template is defined as a pair of a semantic query graph g and its most similar SPARQL query graph q. Based on the derived templates, given a new question, its most similar SPARQL query graph will be found and issued to the KGs to retrieve the answers. In the above process, the semantic query graph is uncertain, i.e., each node/edge has multiple possible labels with different probabilities, because of the semantic ambiguity. For example, the question "which actress from the USA is married to Michael Jordan born in a city of NY" may be mapped to three different "Michael Jordan" and two different "NY". Thus unlike the traditional GED algorithm, Uncertain TQA proposes a common structural subgraph(CSS)-based lower bound to avoid exhaustively enumerating every possible match of g, which is a uniform bound for GED.

Similarly, QUINT proposed by Abujabal et al. [1] translates the question into a dependency parse tree [73]. Instead of matching each question with a SPARQL query graph by UncertainTQA, QUINT retrieves the smallest subgraph connecting the entities in the question and the answers from the KGs, and then maps the graph with the dependency parse tree of the question. The alignment between the two graphs forms a template. QUINT formulates the alignment problem as constrained optimization and finds the best alignment using integer linear programming (ILP).

Both the above two methods generate templates in form of structured graphs. However, unlike a structured graph, a natural language template includes a natural language pattern and a corresponding SPARQL pattern. To generate the natural language template, one should first replace the entities in the question with their types, and then determine the relation type of the template. For example, for the question "When was Barack Obama born?", a natural language pattern "When was $PERSON born?" is derived, which is also mapped to the predicate "BirthPlace" and then is corresponded to the SPARQL pattern "<Person>, birthPlace, ?place". TemplateQA [170] and KBQA [23] are both this kinds of natural language templates. TemplateQA assumes that a natural language pattern is likely to be matched to a predicate if they are simultaneously shared by many entity pairs in KGs. KBQA proposes a probabilistic method to capture the matching likelihood between a natural language pattern and a predicate, where each natural language pattern s is modeled as a hidden variable, and the maximum likelihood is adopted to estimate the matching probability $P(r|s)$.

Given a set of derived templates, TemplateQA computes the Jaccard similarity coefficient between the question q and the natural language template t. For complex questions, TemplateQA builds a semantic dependency graph (SDG) through matching each subsequence of q with templates one by one. When one template is identified, the subsequence is removed and replaced with the answer type of the template. Thus during the decomposition, the type constraint between the two adjacent templates (i.e., the neighboring nodes in SDG) is naturally guaranteed. To reduce the search space, TemplateQA employs both type-based and order-based optimizations. The whole process of TemplateQA is illustrated in Figure 10, which includes the offline templates generation based on the text corpus and the KGs, and the online SDG parsing for the input question based on the generated templates. Once the SDG is parsed, it is mapped to a SPARQL query, which is issued to the KGs.

The above methods deal with KGQA with complex questions in quite different approaches. For example, QUINT defines some dependency parse rewriting rules to get two separate propositions when facing relative clauses and coordinating conjunctions and then connect the two propositions with for example $conj$ or $pobj$ edges in the dependency parse tree, which enables it to answer conjunctive questions. KBQA decomposes the question into a chain of binary sub-questions where the answer of the last sub-question fills in the value of the variable in the next sub-question, which enables it to answer multi-hop questions. However, QUINT depends on the predefined rewriting rules and KBQA only supports multi-hop questions. Thus, compared with them, in TemplateQA, the sub-questions form a dependency graph, which is more expressive. In addition, [89], [105], [158] are similar works which generate natural language templates while [138], [137] generate structured graph templates.

Summary. Semantic parsing and template-based

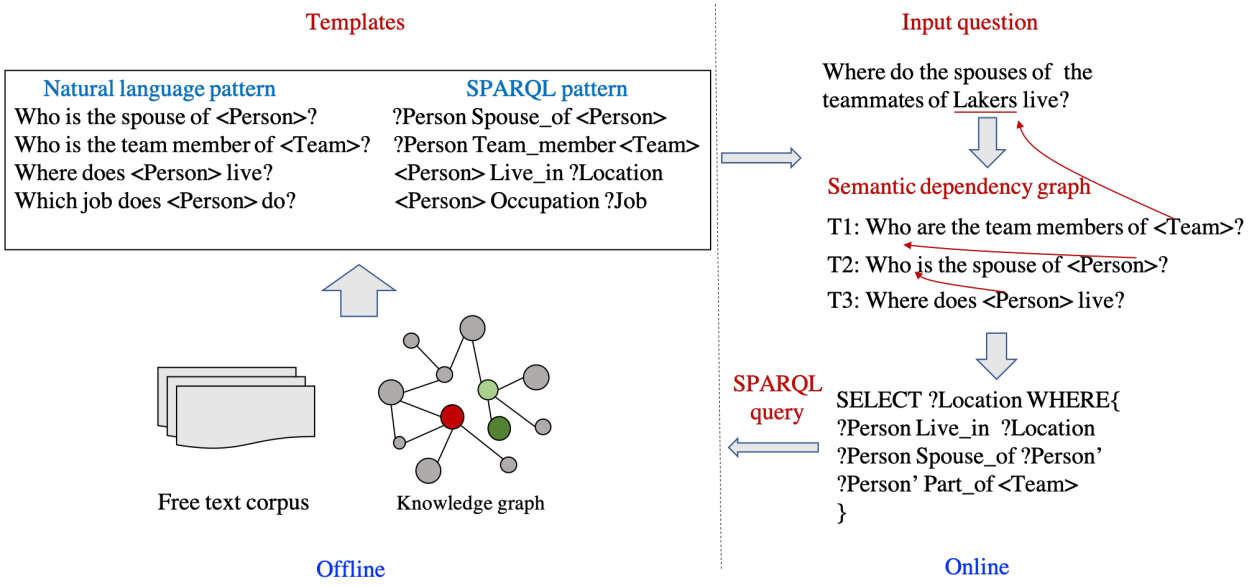


Fig. 10. The framework of TemplateQA [170].

parsing are both symbolic ways to represent a question as a structured query. Semantic parsing directly learns a function to map the question to a structured query, while template-based parsing first generates a collection of templates and then maps the question to an existing template to obtain the structured query. These symbolic reasoning methods are good at dealing with complex logic questions, as the structured query can be trees or graphs, which are expressive enough to represent complex logic questions. However, similar to KGC, the symbolic reasoning methods cannot deal with the ambiguity of natural languages and the uncertainty of entities and relations.

4.3 Neural-Symbolic Reasoning

Neural-symbolic reasoning combines the advantages of both the neural reasoning and the symbolic reasoning for KGQA. These hybrid methods generally fall into two categories: the neural-enhanced symbolic reasoning which only targets at parsing the questions, or the end-to-end reasoning which parses the questions and retrieves the answers simultaneously.

4.3.1 Neural-enhanced Symbolic Reasoning

Neural-enhanced symbolic reasoning still parses the given question into a query graph. After parsing, a neural network is leveraged to evaluate whether the parsed query graph is relevant to the given question.

Single-relation Question. Yih et al. [164] solve the single-relation question by completing two tasks: linking the mention in the question to an entity in KGs as the topic entity, and mapping the relation pattern described by the question to a relation in KGs. For example, given a question such as "*When were DVD players invented?*", the paper first extracts the mention "*DVD players*" and derives the relation pattern "when were X invented", and then links "*DVD players*" to the entity "*dvd-player*" and also maps "when were X invented" to the relation "*be-invent-in*". Once the topic entity and the relation are detected, the answer to the question can be directly queried by the relation-entity triple "*be-invent-in*(*dvd-player*,?)" in the KGs. To extract the mention and derive the relation pattern, the authors simply enumerate all the combinations. The key point is to determine the mapping between the extracted mention and the entity as well as the mapping between the derived pattern and the relation. To achieve the goal, the authors propose a CNN model to take the sequential tokens in a mention or a relation pattern as input and output the corresponding embedding, which is then compared with the embedding of the ground truth entity or relation in KGs.

Multi-hop Question. In comparison with the simple questions, the searching space grows exponentially if the given question involved multi-hop relations. To tackle the multi-hop relation questions, instead of generating the whole query graph at once, MULTIQUE [7] breaks the original question into simple partial queries and builds sub query graphs for partial queries one by one. The search space is shrinked since each time when extending the whole query graph by a new sub-query graph, the model only needs to consider the immediate answers queried by the previous most matched sub-query graph. In the graph generation process, how to measure the quality of a sub-query graph is a key problem to be solved. The authors apply an LSTM model to encode the token sequence of the given question, where an attention mechanism is incorporated to emphasize a particular part of the

TABLE 5
Constraint categories proposed by Bao et al. [2].

Constraint Category	Example	Percentage
Multi-Entity	Which films star by **Forest Whitaker** and are directed by **Mark Rydell**?	30.6%
Type	Which **city** did Bill Clinton born?	38.8%
Explicit Temporal	Who is the governor of Kentucky **2012** ?	10.4%
Implicit Temporal	Who is the us president **when the Civil War started**?	3.5%
Ordinal	What is the **second longest** river in China?	5.1%
Aggregation	**How many** children does Bill Gates have?	1.2%

question which is expected to be represented by the current sub-query graph. Meanwhile, the relation and the constraint in each sub-graph are also encoded by the identification and the tokens. Once the embeddings of the original question and the sub-query graph are encoded, they are concatenated and fed to an MLP, which outputs a scalar to reflect the similarity between the question and the sub-query graph.

Complex-logic Question. Bao et al. [2] further deal with questions with multiple constraints, which are presented in Table 5. They first transform a question into a multi-constraint query graph, then propose a Siamese convolutional neural networks to calculate the similarity between the query graph and the input natural language question, and finally execute the top-ranked query graph on KGs by instantiating all variable nodes according to the constraints in order. The multi-constraint query graph contains two types of nodes, where a constant node represents a ground entity in KGs such as "Barachk Obama" or an attribute value such as "1961", and a variable node represents an unknown entity or unknown attribute value. The graph also contains two types of edges where a relational edge represents a relation such as "birthday" in KGs and a functional edge represents a functional predicate of a truth such as $<$ in the truth $\langle 2000, <, 2001\rangle$. To construct the query graph, an entity linking method proposed by [161] is leveraged to detect topic entities from the given question. For each topic entity, one-hop relations or two-hop relations are extended from it to form a basic query graph, and then different types of constraints in Table 5 are detected from the question and binded to the basic query graph. Later, instead of adding constraints only after relation paths have been constructed, Lan et al. [77] propose a method to incorporate constraints and extend relation paths simultaneously, which can effectively reduce the search space.

4.3.2 End-to-end Reasoning

End-to-end reasoning parses the questions and retrieves the answers in a unified model. According to the form of the connections between the topic entities and the answers, we categorize the end-to-end reasoning into path-based and graph-based methods.

Path-based Reasoning. Path-based methods employ a hop-by-hop path search over KGs, which usually contains three stages of dealing with the input question, reasoning over the KGs and predicting the answers.

For example, IRN (Interpretable Reasoning Network) [172] proposes three modules corresponding to the above three stages in a unified model. The input module initializes the question by the embeddings of the words in the question and updates the question's embedding hop-by-hop according to inference results of the reasoning module. The reasoning module initializes its state by the topic entity of the question and predicts the embedding of the relation $\hat{\mathbf{r}}^h$ at the h-th hop based on the question's embedding $\mathbf{q}^{h-1}$ and the state vector $\mathbf{s}^{h-1}$ at the $(h-1)$-th hop, i.e.,

$$\begin{aligned} g_j^h &= \text{softmax}((\mathbf{M}_{rq}\mathbf{r}_j)^T\mathbf{q}^{h-1}+(\mathbf{M}_{rs}\mathbf{r}_j)^T\mathbf{s}^{h-1}), \\ \hat{\mathbf{r}}^h &= \sum_j g_j^h * \mathbf{r}_j, \end{aligned} \tag{30}$$

where $\mathbf{r}_j$ is the embeding of the j-th relation in KGs, g_j^h is the probability of selecting the j-th relation in KGs, and $\mathbf{M}_{rs}, \mathbf{M}_{rq}$ are the project matrices mapping $\mathbf{r}$ from the relation space to the state space and to the question space respectively. Then the predicted relation $\hat{\mathbf{r}}^h$ is utilized to update the state vector and the question embedding by:

$$\begin{aligned} \mathbf{s}^h &= \mathbf{s}^{h-1} + \mathbf{M}_{rs}\hat{\mathbf{r}}^h, \\ \mathbf{q}^h &= \mathbf{q}^{h-1} - \mathbf{M}_{rq}\hat{\mathbf{r}}^h. \end{aligned} \tag{31}$$

In the above equation, in order to pay attention to different parts of the question at each hop, the predicted relations from previous hops are removed from the question at each hop. The answer module predicts an entity conditioned on the similarity between its pre-trained embedding and the reasoning state at the last hop. In addition to the answer entity, the entities at the intermediate hops are also predicted and evaluated to improve the answer prediction performance, i.e.,

$$\begin{aligned} \mathbf{e}^h &= \mathbf{M}_{se}\mathbf{s}^h, \\ o_j^h &= P(a^h = e_j|\mathbf{s}^h) = softmax(\mathbf{e}_j^T\mathbf{e}^h), \end{aligned} \tag{32}$$

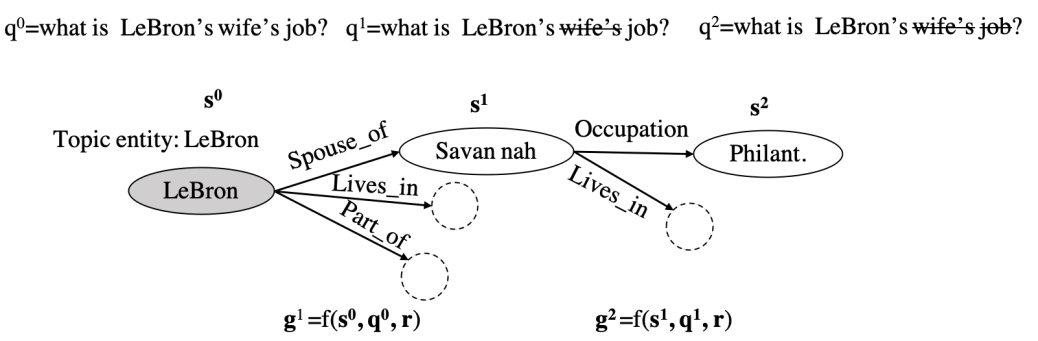


Fig. 11. Illustration of IRN [172]. Notation $\mathbf{s}^h$ represents the state vector of the h-th hop, which is initialized by the embedding of the topic entity. $\mathbf{g}^h$ is the vector of the probabilities to select each relation according to the h-th state vector, question embedding and each relation embedding. The question embedding is updated after each hop's selection.

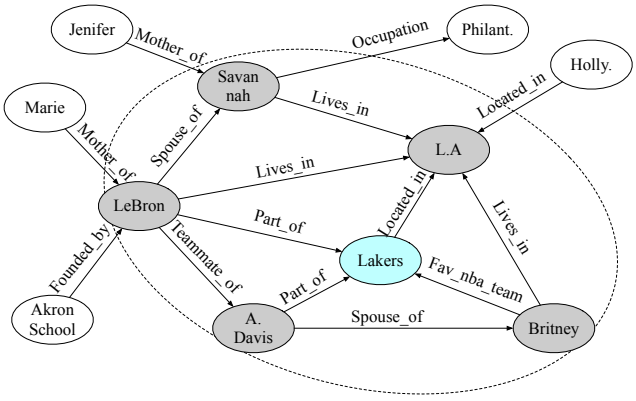
where $\mathbf{M}_{se}$ is the projection matrix mapping $\mathbf{s}^h$ from the state space to the entity space and $\mathbf{e}_j$ is the pre-trained embedding of the j-th entity in KGs. Figure 11 illustrates the three modules in IRN.

IRN assumes that the ground truth paths for answering the questions are also observed, thus the intermediate entities and the answer entities can both supervise the model training process. However, in most cases, we can only obtain the answers to the questions, without knowing the reasoning paths. To deal with the challenge, SRN (Stepwise Reasoning Network) [110] formulates the reasoning process as a Markov decision process where the answer entity can provide the delayed reward to the decision of the relation at each hop. In addition to the delayed reward, SRN also incorporates the intermediate reward to overcome the delayed and sparse rewards. To emphasize different parts of a question, instead of removing the predicted relations from the question by IRN, SRN employs the attention mechanism to decide which part should be focused on at present. [96] [26] [27] are similar path-based works.

The path-based reasoning methods can obtain explicit paths, i.e., logic rules for solving multi-hop QA. These methods usually employ deliberate analysis on the input question in order to focus on different parts of the question at each hop. Meanwhile, since there is a need for reasoning over several hops, intermediate reward can be powerful signals to supervise the whole training process.

Graph-based Reasoning. Instead of extending a single path from the topic entity to the answer, the graph-based reasoning methods extend a subgraph around the topic entity, which is more expressive than a single path. The general idea of graph-based reasoning is to extend a subgraph and then reason the answer in it by the recent technique of graph representation learning.

Graft-Net [127] is a representative graph-based reasoning model. It first performs a Personalized PageRank (PPR) [55] around the topic entities in the question



Question: Where do the spouses of the team members of Lakers usually live?

Fig. 12. Illustration of Graft-Net [127]. Given the topic entity "Lakers" in a question, Graft-Net first extracts the subgraph around the topic entity and then performs a variant GNN model to represent each node in the subgraph.

and then incorporates other entities that might be an answer to the question. In PPR, the relations in KGs which are more relevant to the question are weighted higher. After running PPR, they retain the top entities by PPR score, along with all the relations between them. Graft-Net not only leverages the original KGs, but also incorporates Wikipedia text copus as the additional source to infer the answers. When extracting a subgraph from the text corpus, both the most relevant sentences to the question and any entities linked to these sentences are extracted as the nodes, and then the relations from the KGs among these entities, plus the mention links between the sentences and the entities are extracted as the edges in the subgraph. The subgraphs extracted from the KGs and the text corpus are merged together to construct the final subgraph.

Since the final subgraph is a heterogeneous graph that includes both the entities and the sentences as the

nodes, Graft-Net performs a variant GNN model to represent different types of nodes by different update rules, i.e., the embedding of an entity v at the l-th step is updated as follows:

$$h_v^{(l)} = \text{FFN}\left(\begin{bmatrix} h_v^{(l-1)} \\ h_q^{(l-1)} \\ \sum_r \sum_{v' \in N_r(v)} \alpha_r^{v'} \psi_r(h_{v'}^{(l-1)}) \\ \sum_{(d,p) \in M(v)} H_{d,p}^{(l-1)} \end{bmatrix}\right) \quad (33)$$

where the first two terms correspond to the entity embedding and the question embedding, respectively, from the previous layer. The third term aggregates the states from the entity neighbors of any relations of the current entity, i.e. $N_r(v)$, after scaling with an attention weight $\alpha_r^{v'}$ and applying relation specific transformation ψ_r proposed in R-GCN [120]. The last term aggregates the states of all tokens that correspond to the mentions of the entity v among the documents in the subgraph. In the above equation, $M(v) = \{(d,p)\}$ is the set of document-position pairs, where each pair (d,p) indicates d mentions entity v at the position p. The embedding of a document d in a subgraph at the l-th step is updated by:

$$\begin{aligned} \tilde{H}_{d,p}^{(l)} &= \text{FFN}(H_{d,p}^{(l-1)}, \sum_{v \in L(d,p)} h_v^{(l-1)}) \quad (34) \\ H_d^{(l)} &= \text{LSTM}(\tilde{H}_{d,p}^{(l)}) \end{aligned}$$

where $L(v) = \{(d,p)\}$ is the set of entities linked to entity v at position p at document d. The above equations first aggregate over the entity states coming in at each position separately and then aggregate states within the document using an LSTM network. Graft-Net predicts whether one entity in the subgraph is the answer based on the entity's embedding of the last step. Figure 12 illustrates the basic idea of Graft-Net without the text information.

However, the question-specific subgraphs the Graft-Net builds heuristically are far from optimal, i.e., they are often much larger than necessary, and sometimes do not contain the correct answer. Thus, PullNet [128] proposes a policy function to learn how to construct the subgraph, rather than using an ad-hoc subgraph-building strategy. The classification model used to predict the answer in the subgraph is the same as Graft-Net. Since the intermediate entities in the subgraph are latent, PullNet proposes a weak supervision method to train the policy function. The general idea is to find all shortest paths between topic entities and the answer entities and mark the entities in such shortest paths as the ground truth intermediate entities.

Graft-Net and PullNet assume the topic entities are given, however, in many cases, only the questions are provided. Thus, VRN (Variational Reasoning Network) [169] models the topic entities as hidden variables and proposes two probabilistic modules in a unified architecture, one for topic entity recognition ($P(y|q)$) and the other for logic reasoning ($P(a|y,q)$), where y indicates a hidden topic entity, q refers to the query and a refers to the answer. The two modules are jointly trained such that they can coordinate with and benefit from each other. However, the reasoning-graph is also an ad-hoc subgraph. Instead of performing PPR to build the subgraph as Graft-net does, VRN performs topological sort within T hops starting from a topic entity y to build the scope $\mathcal{G}_y$. For each potential answer $a \in \mathcal{G}_y$, the reasoning graph $\mathcal{G}_{y \to a}$ is represented as the minimum subgraph that contains all the paths from y to a in $\mathcal{G}_y$. Then VRN proposes a "forward graph embedding" method to embed the reasoning-graph for each answer a recursively using its parents' embeddings. Finally, the similarity between the reasoning-graph embedding and the question representation is calculated as the score to predict the answer. Other similar graph-based works include [9] [61].

Summary. The first kind of the neural-symbolic reasoning for KGQA, i.e., the neural-enhanced symbolic reasoning, leverages the neural networks to evaluate the similarity between the parsed query graph and the natural language question. These methods usually need the ground truth of the query graphs for the questions, which are not easy to be annotated.

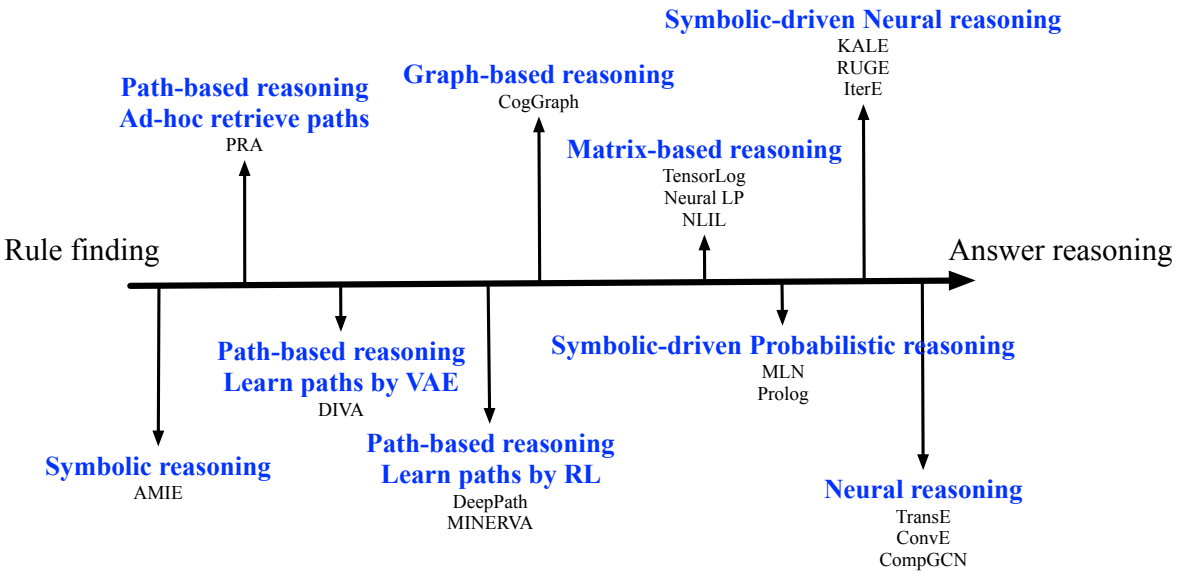
The second kind of the neural-symbolic reasoning for KGQA, i.e., the end-to-end reasoning, directly aligns the questions with the answers, where the answers are easily obtained and can be used as the ground truth to supervise the alignment. The objective of these methods is to represent the inferred path or graph into an embedding vector, based on which the answer can be determined. The reasoned paths/graphs can be viewed as the explanation for the reasoning results. However, the operations over the embeddings can only support the single-relation and multi-hop relation questions. It is still unknown how to address the complex logic questions by the end-to-end reasoning methods.

5 Conclusion and Future Directions

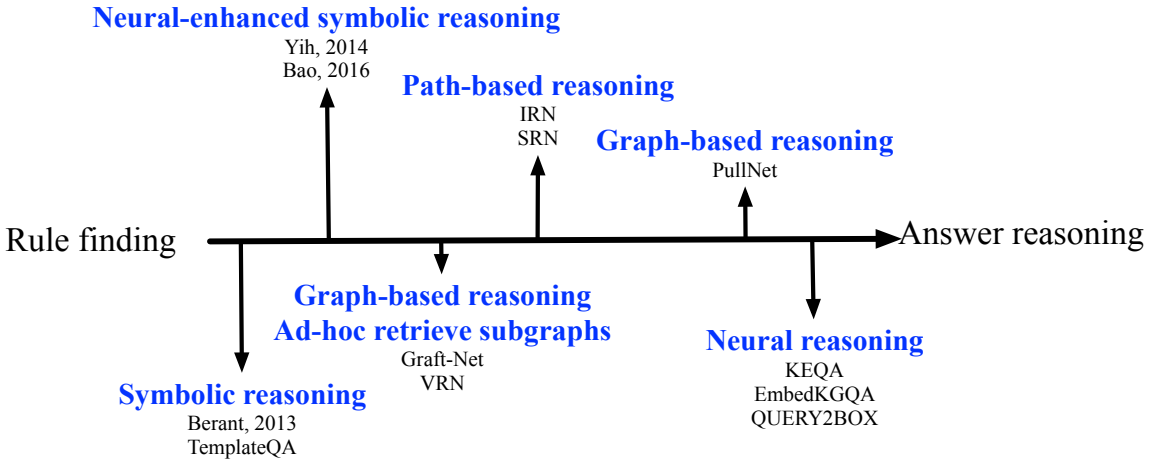
In this section, we first conclude this survey by casting KQC and KGQA in a unified reasoning framework and then discuss some potential future directions.

5.1 Conclusion

Generally, most of the reasoning methods for completion can be decomposed into two key components: rule finding and answer reasoning, where rule finding targets at inferring the rules from the observed triplets in KGs, and answer reasoning aims to predict the answer for the given head entity and the query



(a) KGC



(b) KGQA

Fig. 13. Summary of KGC and KGQA in a unified reasoning framework.

relation. Different methods for KGC emphasize different components. In Figure 13(a), going from left to right, the methods pay more and more attention to answer reasoning and less and less attention on rule finding. For example, on the far left, the pure symbolic reasoning methods such as AMIE only explain how to find rules from the data but without discussing the usage of the rules to reason answers. Then the path-based reasoning method PRA not only ad-hocly retrieves the paths by PageRank, but also trains a simple linear regression model based on the paths to reason answers. However, how to retrieve the paths is still the core problem to be solved in PRA. DIVA treats the two components equally in a unified model through formulating paths as hidden variables. Subsequently, DeepPath and MINERVA directly reason the answers by RL, without explicitly deriving the paths. Graph-based reasoning such as CogGraph extends a single path to a subgraph and matrix-based reasoning such as TensorLog and Neural LP extends the subgraph to the whole graph with attentions on different nodes at each hop. Although the explicit rule finding is missing, the path-based, graph-based, and matrix-based reasoning can still derive the paths according to the selections or attentions at each hop. At the same time, symbolic-driven probabilistic reasoning and symbolic-driven neural reasoning thoroughly take the answer reasoning as the target. The difference is that in the symbolic-driven probabilistic reasoning, the rules are used as the features to predict the answers, while in the symbolic-driven neural reasoning, the rules are used to generate more facts for learning high-quality embeddings. Finally, the neural reasoning methods get rid of the rules and reason

answers entirely based on the embedding techniques.

Most of the reasoning methods for KGQA can also be decomposed into rule finding and answer reasoning, where the former targets at parsing the rule, i.e., a path or a subgraph from the given question, and the latter aims to predict the answer for the given question. In Figure 13(b), the methods from the left to the right gradually pay more attention to answer reasoning and less attention to rule finding. Starting from the left, the pure symbolic methods and the neural-enhanced symbolic reasoning methods only parse the questions to obtain the executable queries, query paths, or subgraphs, where the neural-enhanced symbolic reasoning methods incorporate additional embedding techniques to evaluate the parsed paths or subgraphs. Then the graph-based reasoning methods Graft-Net and VRN retrieve the subgraphs heuristically and then reason the answers from the subgraphs by neural networks or graph convolution networks. Later, the path-based methods IRN and SRN and the graph-based method PullNet reason the paths/subgraphs and the answers in an end-to-end manner by RL or weak supervision techniques. The far right neural reasoning methods get rid of the paths/subgraphs entirely and directly measure the relationships between the answers and the given question. The pure symbolic methods can parse quite complex questions such as those with multiple constraints as they only parse the questions. However, the path/graph-based end-to-end methods need to extend the paths/subgraphs along with the relations in KGs, making it not easy to incorporate the complex logic questions.

5.2 Future Directions

Despite the existing researches on reasoning methods for KGC and KGQA, there are still unsolved challenges on these tasks such as reasoning for the few-shot relations in KGC and reasoning for the complex questions in KGQA. In addition, existing works mainly focus on leveraging the knowledge graph structures for reasoning, but only a few works investigate how to leverage the side information of the entities and the relations to benefit the reasoning task [39], [127], [131], the performance of which can be still improved. Moreover, some other kinds of reasoning tasks on KGs such as dynamic reasoning and analogical reasoning are also worth studying. Finally, existing reasoning methods lack the transferability from one KG to others. We explain these future directions in detail as below.

Few-shot Reasoning. Most of the neural-symbolic models heavily rely on a huge amount of training instances. However, the relationships between entities in KGs are far from complete, especially for the rare relations, making it extremely difficult to capture the underlying patterns of these rare relations.

Few-shot learning is a paradigm proposed for learning in the scenario of lacking training instances, which has first shown the significant performance in computer vision [81]. Then, few-shot learning in KGs, which aims to discover the underlying patterns of a relation with which only a few triplets is associated, has been studied recently [34], [154]. Although these are good attempts, the poor performance of reasoning tasks reported by them indicates that few-shot reasoning is still an unsolved challenge.

Answering Complex Questions. Existing neural-symbolic reasoning models are criticized as most of them can only answer the single-relation questions, or limited-hop relations (e.g., three-hop relations), let alone the questions with additional constraints. Ding et al. [33] showed that the performance of the traditional reasoning models on question answering task decreases dramatically with the increase of the hops, as the search space increases exponentially with the hops, making it difficult to reason the correct answer from such big search space.

Recently for question answering upon text corpus, neural-symbolic models incorporating human cognition have shown their superiority on reasoning capacity [33], as they consistently perform well with the increase of the hops in questions. Thus, a new way is called for to incorporate human cognition into neural-symbolic reasoning in KGs. Humans usually answer multi-hop questions following two reasoning systems, where the first system makes fast and intuitive thinking for collecting enough raw evidence, and the second system makes slow and logical thinking for reasoning among the collected raw data. How to model the two systems in a neural-symbolic reasoning framework in KGs effectively is a promising direction.

Reasoning upon Multi-sources. Since the structure information comprised of entities and relations in a knowledge graph is far from complete, incorporating additional information from unstructured text data for reasoning is encouraged.

Although some models such as Graft-Net [127] and PullNet [128] are the state-of-the-art reasoning models on both the structural and the textual information, it is still challenging to determine the correct evidence to be linked to the incomplete graph structure from the large textual data. Meanwhile, although the textual information can enrich the KGs, it contains much useless and redundant information, which may result in side effects on the reasoning tasks.

Dynamic Reasoning. Dynamic reasoning aims at learning new logic rules and inferring new facts evolving with time. Existing reasoning methods are all devoted to reasoning in the static KGs, but they ignore the temporal information contained in knowledge. However, as we all know, the facts contained in KGs such as (Steve Jobs, CEO of, Apple inc.) are not always true over time. Besides, new knowledge is produced

by humans continually, which may be injected into KGs dynamically. Thus, dynamic reasoning upon the dynamic KGs is demanded to self-correct KGs and mine new logic rules continually.

Analogical Reasoning. Learning quickly is a hallmark of human intelligence, which involves figuring out the underlying patterns in a new domain by adopting the experience in old domains. It will be appreciated that if the reasoning models on KGs are able to perform the same adaptive learning by comparing the similarities between the new KGs and old KGs. We take academic knowledge graphs as examples to explain analogical reasoning. Suppose in these KGs for different academic fields such as computer vision (CV) and natural language processing (NLP), science problems and techniques are added as the nodes, and multiple relationships such as techniques solving problems and techniques citing techniques are added as the edges. Some deep learning techniques such as CNN [74] and pre-training [86] techniques originally proposed in the problems of CV are further adapted to the problems of NLP, and have shown their superior performance, which can be treated as typical analogical reasoning on the problems between CV and NLP.

Knowledge Graph Pre-training. Neural reasoning methods for KGC such as TransE and ConvE reviewed in Section 3.1 produce entity embeddings and relation embeddings, which can be incorporated into the symbolic reasoning process to improve the capacity of fault-tolerance (Cf. Section 3.3.3 for details). However, these neural reasoning models treat the embeddings for all entities and relations in the given knowledge graph as parameters to be learned, which cannot be transferred to other knowledge graphs.

Recently, the transferable pre-training graph neural networks have proved to be able to capture the graph structure characteristics across different graph data [62], [109]. Inspired by the success of the graph pre-training models, a knowledge graph pre-training model that can capture the transferable semantics of the entities and relations across different knowledge graphs is worth studying.

REFERENCES

[1] A. Abujabal, M. Yahya, M. Riedewald, and G. Weikum. Automated template generation for question answering over knowledge graphs. *Proceedings of the 26th International Conference on World Wide Web*, page 11911200, 2017.

[2] J. Bao, N. Duan, Z. Yan, M. Zhou, and T. Zhao. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2503–2514, 2016.

[3] H. Bast and E. Haussmann. More accurate question answering on freebase. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 1431–1440, 2015.

[4] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems*, pages 153–160, 2007.

[5] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1533–1544, 2013.

[6] T. R. Besold, A. d. Garcez, S. Bader, H. Bowman, P. Domingos, P. Hitzler, K.-U. Kühnberger, L. C. Lamb, D. Lowd, P. M. V. Lima, et al. Neural-symbolic learning and reasoning: A survey and interpretation. *arXiv preprint arXiv:1711.03902*, 2017.

[7] N. Bhutani, X. Zheng, K. Qian, Y. Li, and H. Jagadish. Answering complex questions by combining information from curated and extracted knowledge bases. In *Proceedings of the First Workshop on Natural Language Interfaces*, pages 1–10, 2020.

[8] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 1247–1250, 2008.

[9] A. Bordes, S. Chopra, and J. Weston. Question answering with subgraph embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 615–620, 2014.

[10] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems 2013*, pages 2787–2795, 2013.

[11] J. Bos, S. Clark, M. Steedman, J. R. Curran, and J. Hockenmaier. Wide-coverage semantic representations from a CCG parser. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 1240–1246, 2004.

[12] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[13] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[14] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, and T. M. Mitchell. Toward an architecture for never-ending language learning. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, page 13061313, 2010.

[15] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *Proceedings of the 16th International Conference on World Wide Web*, pages 571–580, 2007.

[16] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[17] W. Chen, W. Xiong, X. Yan, and W. Wang. Variational knowledge graph reasoning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1823–1832, 2018.

[18] X. Chen, S. Jia, and Y. Xiang. A review: Knowledge reasoning over knowledge graph. *Expert Systems with Applications*, 141:112948, 2020.

[19] Y. Chen, D. Z. Wang, and S. Goldberg. Scalekb: scalable learning and inference over large knowledge bases. *The VLDB Journal*, 25(6):893–918, 2016.

[20] Y. Chen, L. Wu, and M. J. Zaki. Bidirectional attentive memory networks for question answering over knowledge bases. *ArXiv*, abs/1903.02188, 2019.

[21] P. Clark and T. Niblett. The cn2 induction algorithm. *Machine learning*, 3(4):261–283, 1989.

[22] W. W. Cohen. Tensorlog: A differentiable deductive database. *arXiv preprint arXiv:1605.06523*, 2016.

[23] W. Cui, Y. Xiao, H. Wang, Y. Song, S.-w. Hwang, and W. Wang. Kbqa: Learning question answering over qa corpora and knowledge bases. *Proc. VLDB Endow.*, 10(5):565–576, 2017.

[24] J. Cussens. Parameter estimation in stochastic logic programs. *Machine Learning*, 44(3):245–271, 2001.

[25] Z. Dai, L. Li, and W. Xu. Cfo: Conditional focused neural question answering with large-scale knowledge bases. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

[26] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. Go for a walk and arrive at the answer: Reasoning over knowledge bases with reinforcement learning. In *Proceedings of the 6th Workshop on Automated Knowledge Base Construction*, pages 1–18, 2018.

[27] R. Das, A. Neelakantan, D. Belanger, and A. McCallum. Chains of reasoning over entities, relations, and text using recurrent neural networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics*, pages 132–141, 2017.

[28] L. de Penning, A. Garcez, L. C. Lamb, and J. Meyer. A neural-symbolic cognitive agent for online learning and reasoning. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, volume 2, pages 1653–1658, 2011.

[29] L. De Raedt, A. Kimmig, and H. Toivonen. Problog: A probabilistic prolog and its application in link discovery. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 2462–2467. Hyderabad, 2007.

[30] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[31] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[32] B. Dhingra, M. Zaheer, V. Balachandran, G. Neubig, R. Salakhutdinov, and W. Cohen. Differentiable reasoning over a virtual knowledge base. *ArXiv*, abs/2002.10640, 2020.

[33] M. Ding, C. Zhou, Q. Chen, H. Yang, and J. Tang. Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2694–2703, 2019.

[34] Z. Du, C. Zhou, M. Ding, H. Yang, and J. Tang. Cognitive knowledge graph reasoning for one-shot relational learning. In *https://arxiv.org/abs/1906.05489*, 2019.

[35] M. Dubey, S. Dasgupta, A. Sharma, K. Höffner, and J. Lehmann. Asknow: A framework for natural language query formalization in sparql. In *The Semantic Web. Latest Advances and New Domains - 13th International Conference*, pages 300–316, 2016.

[36] R. Evans and E. Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[37] P. A. Flach. *Simply logical - intelligent reasoning by example*. Wiley professional computing. Wiley, 1994.

[38] B. Fu, Y. Qiu, C. Tang, Y. Li, H. Yu, and J. Sun. A survey on complex question answering over knowledge base: Recent advances and challenges. *ArXiv*, abs/2007.13069, 2020.

[39] C. Fu, T. Chen, M. Qu, W. Jin, and X. Ren. Collaborative policy learning for open knowledge graph reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 2672–2681, 2019.

[40] N. Fuhr. Probabilistic dataloga logic for powerful retrieval methods. In *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 282–290, 1995.

[41] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek. Fast rule mining in ontological knowledge bases with amie. *The VLDB Journal*, 24(6):707–730, 2015.

[42] L. A. Galárraga, C. Teflioudi, K. Hose, and F. Suchanek. Amie: association rule mining under incomplete evidence in ontological knowledge bases. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 413–422, 2013.

[43] S. I. Gallant. Connectionist expert systems. *Communications of the ACM*, 31(2):152–169, 1988.

[44] S. I. Gallant. *Neural network learning and expert systems*. MIT press, 1993.

[45] A. d. Garcez, M. Gori, L. C. Lamb, L. Serafini, M. Spranger, and S. N. Tran. Neural-symbolic computing: An effective methodology for principled integration of machine learning and reasoning. *arXiv preprint arXiv:1905.06088*, 2019.

[46] A. S. d. Garcez, K. B. Broda, and D. M. Gabbay. *Neural-symbolic learning systems: foundations and applications*. Springer Science & Business Media, 2012.

[47] M. Gardner, P. P. Talukdar, B. Kisiel, and T. Mitchell. Improving learning and inference in a large knowledge-base using latent syntactic cues. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 833–838. Association for Computational Linguistics, 2013.

[48] D. Golub and X. He. Character-level question answering with attention. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1598–1607, 2016.

[49] M. R. Gormley, M. Dredze, and J. Eisner. Approximation-aware dependency parsing by belief propagation. *Transactions of the Association for Computational Linguistics (TACL)*, 3:489–501, 2015.

[50] L. Guo, Z. Sun, and W. Hu. Learning to exploit long-term relational dependencies in knowledge graphs. In *Proceedings of the 36th International Conference on Machine Learning*, pages 2505–2514, 2019.

[51] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. Jointly embedding knowledge graphs and logical rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 192–202, 2016.

[52] S. Guo, Q. Wang, L. Wang, B. Wang, and L. Guo. Knowledge graph embedding with iterative guidance from soft rules. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 4816–4823, 2018.

[53] W. L. Hamilton, P. Bajaj, M. Zitnik, D. Jurafsky, and J. Leskovec. Embedding logical queries on knowledge graphs. In *Advances in Neural Information Processing Systems 31*, pages 2030–2041, 2018.

[54] J. Haugeland. *Artificial intelligence: The very idea*. MIT press, 1989.

[55] T. H. Haveliwala. Topic-sensitive pagerank. In *Proceedings of the Eleventh International World Wide Web Conference*, pages 517–526, 2002.

[56] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[57] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[58] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[59] F. L. Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Studies in Applied Mathematics*, 6:164189, 1927.

[60] V. T. Ho, D. Stepanova, M. H. Gad-Elrab, E. Kharlamov, and G. Weikum. Rule learning from knowledge graphs guided by embedding models. In *Proceedings of the 17th International Semantic Web Conference*, pages 72–90. Springer, 2018.

[61] S. Hu, L. Zou, and X. Zhang. A state-transition framework to answer complex questions over knowledge base. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2098–2108, 2018.

[62] Z. Hu, Y. Dong, K. Wang, K.-W. Chang, and Y. Sun. Gpt-gnn: Generative pre-training of graph neural networks. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1857–1867, 2020.

[63] X. Huang, J. Zhang, D. Li, and P. Li. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 105–113, 2019.

[64] P. Hajek. The metamathematics of fuzzy logic. 1998.

[65] A. Jha and D. Suciu. Probabilistic databases with markoviews. *Proceedings of the VLDB Endowment*, 5(11), 2012.

[66] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao. Knowledge graph embedding via dynamic mapping matrix. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 687–696, 2015.

[67] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S. Yu. A survey on knowledge graphs: Representation, acquisition and applications. In *arXiv preprint arXiv:2002.00388*, 2020.

[68] X. Jiang, Q. Wang, and B. Wang. Adaptive convolution for multi-relational learning. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 978–987, 2019.

[69] R. J. Kate, Y. W. Wong, and R. J. Mooney. Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1062–1068, 2005.

[70] S. M. Kazemi and D. Poole. Simple embedding for link prediction in knowledge graphs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 4284–4295. 2018.

[71] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[72] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[73] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, page 423430, 2003.

[74] A. Krizhevsky, I. Sutskever, and E. G. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, pages 84–90, 2017.

[75] T. Kwiatkowksi, L. Zettlemoyer, S. Goldwater, and M. Steedman. Inducing probabilistic ccg grammars from logical form with higher-order unification. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1223–1233, 2010.

[76] L. Lamb, A. Garcez, M. Gori, M. Prates, P. Avelar, and M. Vardi. Graph neural networks meet neural-symbolic computing: A survey and perspective. *arXiv preprint arXiv:2003.00330*, 2020.

[77] Y. Lan and J. Jiang. Query graph generation for answering multi-hop complex questions from knowledge bases. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 969–974, July 2020.

[78] N. Lao and W. W. Cohen. Relational retrieval using a combination of path-constrained random walks. *Machine learning*, 81(1):53–67, 2010.

[79] N. Lao, A. Subramanya, F. Pereira, and W. W. Cohen. Reading the web with learned syntactic-semantic inference rules. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1017–1026, 2012.

[80] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. Dbpedia: A large-scale, multilingual knowledge base extracted from wikipedia. 6:167–195, 2015.

[81] F. Li, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4):594–611, 2006.

[82] P. Liang. Lambda dependency-based compositional semantics. *Technical report*, 2013.

[83] X. V. Lin, R. Socher, and C. Xiong. Multi-hop knowledge graph reasoning with reward shaping. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3243–3253, 2018.

[84] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2181–2187, 2015.

[85] H. Liu, Y. Wu, and Y. Yang. Analogical inference for multi-relational embeddings. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70, pages 2168–2178, 2017.

[86] X. Liu, F. Zhang, Z. Hou, Z. Wang, L. Min, J. Zhang, and J. Tang. Self-supervised learning: Generative or contrastive. In *https://arxiv.org/abs/2006.08218*, 2020.

[87] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. In *arXiv preprint arXiv:1907.11692*, 2019.

[88] D. Lukovnikov, A. Fischer, J. Lehmann, and S. Auer. Neural network-based question answering over knowledge graphs on word and character level. In *Proceedings of the 26th International Conference on World Wide Web*, pages 1211–1220, 2017.

[89] G. M. Mazzeo and C. Zaniolo. Answering controlled natural language questions on rdf knowledge bases. In *Proceedings of the 19th International Conference on Extending Database Technology*, pages 608–611, 2016.

[90] C. Meilicke, M. W. Chekol, M. Fink, and H. Stuckenschmidt. Reinforced anytime bottom up rule learning for knowledge graph completion. *arXiv preprint arXiv:2004.04412*, 2020.

[91] R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, pages 1041–1045, 1986.

[92] M. Minsky and S. Papert. An introduction to computational geometry. *Cambridge tiass., HIT*, 1969.

[93] S. Mohammed, P. Shi, and J. Lin. Strong baselines for simple question answering over knowledge graphs with and without neural networks. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 291–296, 2018.

[94] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul. Learning attention-based embeddings for relation prediction in knowledge graphs. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, page 47104723, 2019.

[95] R. M. Neal and G. E. Hinton. A view of the em algorithm that justifies incremental, sparse, and other variants. *Learning in graphical models*, page 355368, 1999.

[96] A. Neelakantan, B. Roth, and A. McCallum. Compositional vector space models for knowledge base inference. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, volume 1, pages 156–166, 2015.

[97] T. D. Nguyen, D. Q. Nguyen, D. Phung, et al. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, 2018.

[98] M. Nickel, L. Rosasco, and T. Poggio. Holographic embeddings of knowledge graphs. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, page 19551961, 2016.

[99] M. Nickel, V. Tresp, and H. P. Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, page 809816, 2011.

[100] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *Proceedings of the 28th International Conference on Machine Learning*, volume 11, pages 809–816, 2011.

[101] G. Niu, B. Li, Y. Zhang, Y. Sheng, C. Shi, J. Li, and S. Pu. Joint semantics and data-driven path representation for knowledge graph inference. *ArXiv*, abs/2010.02602, 2020.

[102] P. G. Omran, K. Wang, and Z. Wang. Scalable rule learning via learning representation. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 2149–2155, 2018.

[103] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[104] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[105] W. T. Pedoe. True knowledge: Open-domain question answering using structured knowledge and inference. *AI Mag.*, 31(3):80–92, 2010.

[106] G. D. Plotkin. A note on inductive generalization. *Machine Intelligence*, 5(1):153–163, 1970.

[107] G. D. Plotkin. A further note on inductive generalization. *Machine Intelligence*, 6(101-124):248, 1971.

[108] W. Qian, C. Fu, Y. Zhu, D. Cai, and X. He. Translating embeddings for knowledge graph completion with relation attention mechanism. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pages 4286–4292, 2018.

[109] J. Qiu, Q. Chen, Y. Dong, J. Zhang, H. Yang, M. Ding, K. Wang, and J. Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2020.

[110] Y. Qiu, Y. Wang, X. Jin, and K. Zhang. Stepwise reasoning for multi-relation question answering over knowledge graph with weak supervision. In *Proceedings of The Thirteenth ACM International Conference on Web Search and Data Mining*, pages 474–482, 2020.

[111] M. Qu and J. Tang. Probabilistic logic neural networks for reasoning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019*, pages 7712–7722, 2019.

[112] H. Ren, W. Hu, and J. Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

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

[114] S. Riedel, L. Yao, and A. McCallum. Modeling relations and their mentions without labeled text. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer, 2010.

[115] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[116] A. Rossi, D. Firmani, A. Matinata, P. Merialdo, and D. Barbosa. Knowledge graph embedding for link prediction: A comparative analysis. In *arXiv preprint arXiv:2002.00819*, 2020.

[117] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[118] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 38593869, 2017.

[119] A. Saxena, A. Tripathi, and P. Talukdar. Improving multi-hop question answering over knowledge graphs using knowledge base embeddings. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4498–4507, 2020.

[120] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pages 593–607, 2018.

[121] M. Schmitz, S. Soderland, R. Bart, O. Etzioni, et al. Open language learning for information extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534, 2012.

[122] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou. End-to-end structure-aware convolutional networks for knowledge base completion. In *Proceedings of the Thirty-third AAAI Conference on Artificial Intelligence*, 2019.

[123] J. W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An experimental comparison. *Machine learning*, 6(2):111–143, 1991.

[124] Y. Shen, J. Chen, P.-S. Huang, Y. Guo, and J. Gao. M-walk: Learning to walk in graph with monte carlo tree search. In *Proceedings of 6th International Conference on Learning Representations*, pages 6686–6797, 2018.

[125] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A core of semantic knowledge. In *Proceedings of the 16th International Conference World Wide Web*, pages 697–706, 2007.

[126] H. Sun, A. O. Arnold, T. Bedrax-Weiss, F. Pereira, and W. Cohen. Faithful embeddings for knowledge base queries. In *Advances in Neural Information Processing Systems*, 2020.

[127] H. Sun, B. Dhingra, M. Zaheer, K. Mazaitis, R. Salakhutdinov, and W. Cohen. Open domain question answering using early fusion of knowledge bases and text. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4231–4242, 2018.

[128] H. Sun, T. B. Weiss, and W. W. Cohen. Pullnet: Open domain question answering with iterative retrieval on knowledge bases and text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 2380–2390, 2019.

[129] Z. Sun, Z. Deng, J. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

[130] K. K. Teru, E. Denis, and W. L. Hamilton. Inductive relation prediction by subgraph reasoning. *arXiv*, 2019.

[131] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2015.

[132] G. G. Towell and J. W. Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1-2):119–165, 1994.

[133] G. G. Towell and J. W. Shavlik. Refining symbolic knowledge using neural networks. *Machine learning: A multistrategy approach*, 4:405–429, 1994.

[134] T. Trouillon, J. Welbl, S. Riedel, E. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 2071–2080, 2016.

[135] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard. Complex embeddings for simple link prediction. *Proceedings of the 33nd International Conference on Machine Learning*, 48:2071–2080, 2016.

[136] F. Türe and O. Jojic. No need to pay attention: Simple recurrent neural networks work! In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2866–2872. Association for Computational Linguistics, 2017.

[137] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In *Proceedings of the 21st International Conference on World Wide Web*, page 639648, 2012.

[138] C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *Proceedings of the 16th International Conference on Natural Language Processing and Information Systems*, page 153160, 2011.

[139] S. Vashishth, S. Sanyal, V. Nitin, N. Agrawal, and P. P. Talukdar. Interacte: Improving convolution-based knowledge graph embeddings by increasing feature interactions. In *AAAI*, pages 3009–3016, 2020.

[140] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks. *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[141] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[142] T. Vu, T. D. Nguyen, D. Q. Nguyen, D. Phung, et al. A capsule network-based embedding model for knowledge graph completion and search personalization. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2180–2189, 2019.

[143] P. Wang, D. Dou, F. Wu, N. d. Silva, and L. Jin. Logic rules powered knowledge graph embedding. *ArXiv*, abs/1903.03772, 2019.

[144] P. Wang, D. Stepanova, C. Domokos, and J. Z. Kolter. Differentiable learning of numerical rules in knowledge graphs. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[145] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29:2724–2743, 2017.

[146] W. Y. Wang, K. Mazaitis, and W. W. Cohen. Programming with personalized pagerank: a locally groundable first-order probabilistic logic. In *Proceedings of the 22nd ACM International Conference on Information & Knowledge Management*, pages 2129–2138, 2013.

[147] Z. Wang, Z. Ren, C. He, P. Zhang, and Y. Hu. Robust embedding with multi-level structures for link prediction. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5240–5246, 2019.

[148] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, page 11121119, 2014.

[149] Y. W. Wong and R. Mooney. Learning synchronous grammars for semantic parsing with lambda calculus. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 960–967, 2007.

[150] P. Wu, X. Zhang, and Z. Feng. A survey of question answering over knowledge base. In *Knowledge Graph and Semantic Computing: Knowledge Computing and Language Understanding*, pages 86–97, 2019.

[151] H. Xiao, M. Huang, Y. Hao, and X. Zhu. Transg: A generative mixture model for knowledge graph embedding. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2316–2325, 2016.

[152] X. Xiaoran, F. Wei, J. Yunsheng, X. Xiaohui, S. Zhiqing, and D. Zhi-Hong. Dynamically pruned message passing networks for large-scale knowledge graph reasoning. In *In Eighth International Conference on Learning Representations*, 2020.

[153] W. Xiong, T. Hoang, and W. Y. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573, 2017.

[154] W. Xiong, M. Yu, S. Chang, X. Guo, and W. Y. Wang. One-shot relational learning for knowledge graphs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1980–1990, 2018.

[155] C. Xu and R. Li. Relation embedding with dihedral group in knowledge graph. In *Proceedings of the 57th Conference of the Association for Computational Linguistics*, pages 263–272, 2019.

[156] K. Xu, S. Zhang, Y. Feng, and D. Zhao. Answering natural language questions via phrasal semantic parsing. In *CCF International Conference on Natural Language Processing and Chinese Computing*, pages 333–344. Springer, 2014.

[157] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390, 2012.

[158] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, page 379390, 2012.

[159] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[160] F. Yang, Z. Yang, and W. W. Cohen. Differentiable learning of logical rules for knowledge base reasoning. In *Advances in Neural Information Processing Systems 30*, pages 2319–2328, 2017.

[161] Y. Yang and M. Chang. S-MART: Novel tree-based structured learning algorithms applied to tweet entity linking. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 504–513, 2015.

[162] Y. Yang and L. Song. Learn to explain efficiently via neural logic inductive learning. In *Proceedings of the 8th International Conference on Learning Representations*, 2020.

[163] R. Ye, X. Li, Y. Fang, H. Zang, and M. Wang. A vectorized relational graph convolutional network for multi-relational network alignment. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 4135–4141, 2019.

[164] W. Yih, X. He, and C. Meek. Semantic parsing for single-relation question answering. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 643–648, 2014.

[165] W. Yin, M. Yu, B. Xiang, B. Zhou, and H. Schütze. Simple question answering by attentive convolutional neural network. In *Proceedings of the 26th International Conference on Computational Linguistics*, pages 1746–1756, 2016.

[166] Y. You, T. Chen, Y. Sui, T. Chen, Z. Wang, and Y. Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33, 2020.

[167] L. S. Zettlemoyer and M. Collins. Learning context-dependent mappings from sentences to logical form. In *Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics*, pages 976–984, 2009.

[168] W. Zhang, B. Paudel, L. Wang, J. Chen, H. Zhu, W. Zhang, A. Bernstein, and H. Chen. Iteratively learning embeddings and rules for knowledge graph reasoning. In *Proceedings of The World Wide Web Conference, 2019*, pages 2366–2377, 2019.

[169] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, and L. Song. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 6069–6076, 2018.

[170] W. Zheng, J. X. Yu, L. Zou, and H. Cheng. Question answering over knowledge graphs: Question understanding via template decomposition. *Proceedings of the VLDB Endowment*, 11(11):1373–1386, 2018.

[171] W. Zheng, L. Zou, X. Lian, J. X. Yu, S. Song, and D. Zhao. How to build templates for rdf question/answering: An uncertain graph similarity join approach. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1809–1824, 2015.

[172] M. Zhou, M. Huang, and X. Zhu. An interpretable reasoning network for multi-relation question answering. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2010–2022, 2018.

[173] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 313–324, 2014.

[174] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over rdf: a graph data driven approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 313–324, 2014.