Knowledge Base Question Answering via Encoding of Complex Query Graphs

Kangqi Luo[1] **Fengli Lin**[1] **Xusheng Luo**[2] **Kenny Q. Zhu**[1]
[1]Shanghai Jiao Tong University, Shanghai, China
[2]Alibaba Group, Hangzhou, China
{luokangqi,fenglilin}@sjtu.edu.cn, lxs140564@alibaba-inc.com, kzhu@cs.sjtu.edu.cn

Abstract

Answering complex questions that involve multiple entities and multiple relations using a standard knowledge base is an open and challenging task. Most existing KBQA approaches focus on simpler questions and do not work very well on complex questions because they were not able to simultaneously represent the question and the corresponding complex query structure. In this work, we encode such complex query structure into a uniform vector representation, and thus successfully capture the interactions between individual semantic components within a complex question. This approach consistently outperforms existing methods on complex questions while staying competitive on simple questions.

1 Introduction

The knowledge-based question answering (KBQA) is a task which takes a natural language question as input and returns a factual answer using structured knowledge bases such as Freebase (Bollacker et al., 2008), YAGO (Suchanek et al., 2007) and DBpedia (Auer et al., 2007). One simple example is a question like this: "What's the capital of the United States?" A common answer to such question is to identify the focus entity and the main relation predicate (or a sequence) in the question, and map the question to a triple fact query (US, $capital$, ?) over KB. The object answers are returned by executing the query. The mapping above is typically learned from question-answer pairs through distant supervision.

While the above question can be answered by querying a single predicate or predicate sequence in the KB, many other more complex questions cannot, e.g. the question in Figure 1. To answer the question "What is the second longest river in United States", we need to infer several semantic clues: 1) the answer is contained by United States; 2) the answer is a river; 3) the answer ranks second by its length in descending order. Thus, multiple predicates are required to constrain the answer set, and we call such questions "complex questions" throughout this paper.

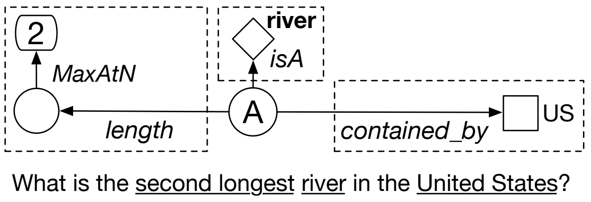

Figure 1: Running example of complex question.

For answering complex questions, it's more important to understand the compositional semantic meanings of the question. As a classic branch of KBQA solutions, semantic parsing (SP) technique (Berant et al., 2013; Yih et al., 2015; Reddy et al., 2016; Hu et al., 2018) aims at learning semantic parse trees or equivalent query graphs [1] for representing semantic structures of the questions. For example in Figure 1, the query graph forms a tree shape. The answer node A, serving as the root of the tree, is the variable vertex that represents the real answer entities. The focus nodes (*US*, *river*, *2nd*) are extracted from the mentions of the question, and they constrain the answer node via predicate sequences in the knowledge base. Recently, neural network (NN) models have shown great promise in improving the performance of KBQA systems, and SP+NN techniques become the state-of-the-art on several KBQA datasets (Qu et al., 2018; Bao et al., 2016). According to the discussion above, our work extends the current research in the SP+NN direction.

The common step of SP-based approaches

[1]The term "query graph" is interchangeable with "query structure" and "semantic parsing tree" throughout this paper.

is to first collect candidate query graphs using bottom up parsing (Berant et al., 2013; Cai and Yates, 2013) or staged query generation methods (Yih et al., 2015; Bao et al., 2016), then predict the best graph mainly based on the semantic similarity with the given question. Existing NN-based methods follow an encode-and-compare framework for answering simple questions, where both the question and the predicate sequence are encoded as semantic vectors in a common embedding space, and the semantic similarity is calculated by the cosine score between vectors. In order to define the similarity function between one question and a complex query graph, an intuitive solution is to split the query graph into multiple semantic components, as the predicate sequences separated by dashed boxes in Figure 1. Then previous methods can be applied for modeling the similarity between the question and each part of the graph. However, such approach faces two limitations. First, each semantic component is not directly comparable with the whole question, since it conveys only partial information of the question. Second, and more importantly, the model encodes different components separately, without learning the representation of the whole graph, hence it's not able to capture the compositional semantics in a global perspective.

In order to attack the above limitations, we propose a neural network based approach to improve the performance of semantic similarity measurement in complex question answering. Given candidate query graphs generated from one question, our model embeds the question surface and predicate sequences into a uniform vector space. The main difference between our approach and previous methods is that we integrate hidden vectors of various semantic components and encode their interaction as the hidden semantics of the entire query graph. In addition, to cope with different semantic components of a query graph, we leverage dependency parsing information as a complementary of sentential information for question encoding, which makes the model better align each component to the question. The contribution of this paper is summarized below.

- We propose a light-weighted and effective neural network model to solve complex KBQA task. To the best of our knowledge, this is the first attempt to explicitly encode the complete semantics of a complex query graph (Section 2.2);
- We leverage dependency parsing to enrich question representation in the NN model, and conduct thorough investigations to verify its effectiveness (Section 2.2.2);
- We propose an ensemble method to enrich entity linking from a state-of-the-art linking tool, which further improves the performance of the overall task (Section 2.3);
- We perform comprehensive experiments on multiple QA datasets, and our proposed method consistently outperforms previous approaches on complex questions, and produces competitive results on datasets made up of simple questions (Section 3).

2 Approach

In this section, we present our approach for solving complex KBQA. First, we generate candidate query graphs by staged generation method (Section 2.1). Second, we measure the semantic similarities between the question and each query graph using deep neural networks (Section 2.2). Then we introduce an ensemble approach for entity linking enrichment (Section 2.3), Finally, we discuss the prediction and parameter learning step of this task (Section 2.4).

2.1 Query Graph Generation

We illustrate our staged candidate generation method in this section. Compared to previous methods, such as Bao et al. (2016), we employ a more effective candidate generation strategy, which takes advantage of implicit type information in query graphs and time interval information in the KB. In our work, we take 4 kinds of semantic constraints into account: **entity**, **type**, **time** and **ordinal** constraints. Figure 2 shows a concrete example of our candidate generation. For simplicity of discussion, we assume Freebase as the KB in this section.

Step 1: Focus linking. We extract possible (mention, focus node) pairs from the question. Focus nodes are the starting points of various semantic constraints, refer to Figure 2(a). For entity linking, we generate (mention, entity) pairs using the state-of-the-art entity linking tool S-MART (Yang and Chang, 2015). For type linking, we brutally combine each type with all uni-, bi-

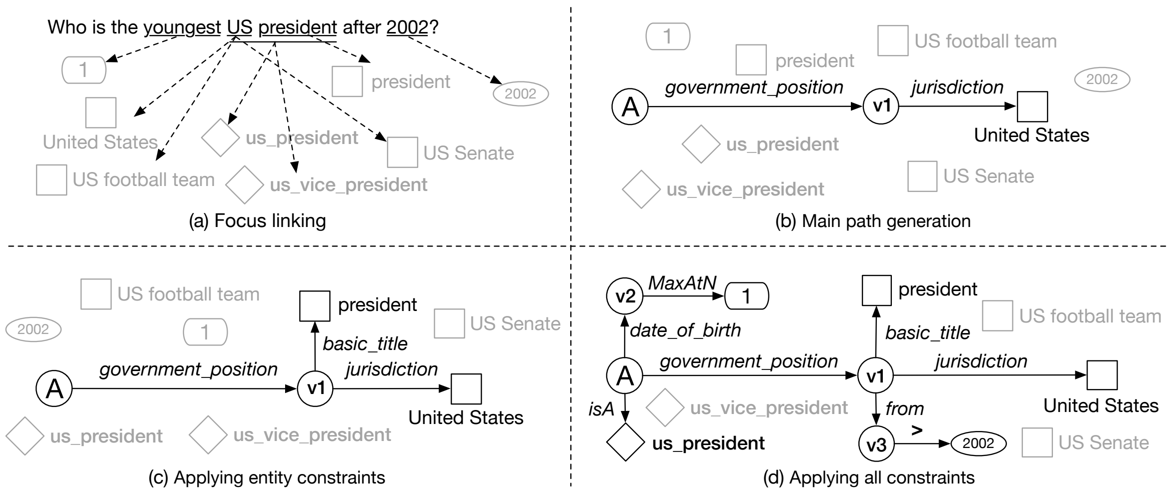

Figure 2: Running example of candidate generation.

and tri-gram mentions in the question, and pick top-10 (mention, type) pairs with the highest word embedding similarities of each pair. For time linking, we extract time mentions by simply matching year regex. For ordinal linking, we leverage a predefined superlative word list[2] and recognize mentions by matching superlative words, or the "ordinal number + superlative" pattern. The ordinal node is an integer representing the ordinal number in the mention.

Step 2: Main path generation. We build different main paths by connecting the answer node to different focus entities using 1-hop or 2-hop-with-mediator[3] predicate sequence. Figure 2(b) shows one of the main paths. Further constraints are attached by connecting an anchor node x to an unused focus node through predicate sequences, where the anchor node x is a non-focus node in the main path (A or v_1 in the example).

Step 3: Attaching entity constraints. We apply a depth-first search to search for combinations of multiple entity constraints to the main path through 1-hop predicate. Figure 2(c) shows a valid entity constraint, $(v_1, basic\_title, president)$. The advantage of depth-first search is that we can involve unlimited number of entities in a query graph, which has a better coverage than template-based methods.

Step 4: Type constraint generation. Type constraints can only be applied at the answer node using *IsA* predicate. Our improvement in this step is to filter type constraints using **implicit types** of the answer, derived from the outgoing predicates of the answer node. For example in Figure 2(c), the domain type of the predicate *government_position* is *politician*, which becomes the implicit type of the answer. Thus we can filter type constraints which are irrelevant to the implicit types, preventing semantic drift and speeding up the generation process. To judge whether two types in Freebase are relevant or not, we adopt the method in Luo et al. (2015) to build a rich type hierarchy of Freebase. Focus types are discarded, if they are not the super- or sub- types of any implicit types of the answer.

Step 5: Time and ordinal constraint generation. As shown in Figure 2(d), the time constraint is represented as a 2-hop predicate sequence, where the second is a virtual predicate determined by the preposition before the focus time, indicating the time comparing operation, like "before", "after" and "in". Similarly, the ordinal constraint also forms a 2-hop predicate sequence, where the second predicate represents descending (*MaxAtN*) or ascending order (*MinAtN*).

For the detail of time constraint, while existing approaches (Yih et al., 2015; Bao et al., 2016) link the focus time with only single time predicate, our improvement is to leverage **paired time predicates** for representing a more accurate time constraint. In Freebase, paired time predicates are used to represent facts within certain time intervals, like $from$ and to[4] in Figure 2(d). For time comparing operation "in", we link the time focus to the starting time predicate, but use both predi-

[2] ~20 superlative words, such as largest, highest, latest.

[3] Mediator is a kind of auxiliary nodes in Freebase maintaining N-ary facts.

[4] Short for $governmental\_position\_held.from$ and $governmental\_position\_held.to$ respectively.

cates in SPARQL query, restricting that the focus time lies in the time interval of the paired predicates.

After finishing all these querying stages, we translate candidate graphs into SPARQL query, and produce their final output answers. Finally, we discard query graphs with zero outputs, or using overlapped mentions.

2.2 NN-based Semantic Matching Model

The architecture of the proposed model is shown in Figure 3. We first replace all entity (or time) mentions used in the query graph by dummy tokens $\langle E \rangle$ (or $\langle Tm \rangle$). To encode the complex query structure, we split it into predicate sequences starting from answer to focus nodes, which we call *semantic components*. The predicate sequence doesn't include the information of focus nodes, except for type constraints, where we append the focus type to the *IsA* predicate, resulting in the predicate sequence like $\{$*IsA*, *river*$\}$. We introduce in detail the encoding methods for questions and predicate sequences, and how to calculate the semantic similarity score.

2.2.1 Semantic Component Representation

To encode a semantic component p, we take the sequence of both predicate ids and predicate names into consideration. As the example shown in Figure 3, the id sequence of the first semantic component is $\{$*contained_by*$\}$, and the predicate word sequence is the concatenation of canonical names for each predicate, that is $\{$“contained”, “by”$\}$.

Given the word sequence $\{p_1^{(w)}, \ldots, p_n^{(w)}\}$, we first use a word embedding matrix $E_w \in \mathbb{R}^{|V_w| \times d}$ to convert the original sequence into word embeddings $\{\boldsymbol{p}_1^{(w)}, \ldots, \boldsymbol{p}_n^{(w)}\}$, where $|V_w|$ denotes the vocabulary size of natural language words, and d denotes the embedding dimension. Then we represent the word sequence using word averaging: $\boldsymbol{p}^{(w)} = \frac{1}{n}\sum_i \boldsymbol{p}_i^{(w)}$.

For the id sequence $\{p_1^{(id)}, \ldots, p_m^{(id)}\}$, we simply take it as a whole unit, and directly translate it into vector representation using the embedding matrix $E_p \in \mathbb{R}^{|V_p \times d|}$ at path level, where $|V_p|$ is the vocabulary size of predicate sequences. There are two reasons for using such path embedding: 1) the length of id sequence is not larger than two, based on our generation method; 2) the number of distinct predicate sequences is roughly the same as the number of distinct predicates. We get the final vector of the semantic component by element-wise addition: $\boldsymbol{p} = \boldsymbol{p}^{(w)} + \boldsymbol{p}^{(id)}$.

2.2.2 Question Representation

We encode the question in both global and local level, which captures the semantic information with respect to each component p.

The global information takes the token sequence as the input. We use the same word embedding matrix E_w to convert the token sequence into vectors $\{\boldsymbol{q}_1^{(w)}, \ldots, \boldsymbol{q}_n^{(w)}\}$. Then we encode the token sequence by applying bidirectional GRU network (Cho et al., 2014). The representation of the token sequence is the concatenation of the last forward and backward hidden states through the Bi-GRU layer, $\boldsymbol{q}^{(tok)} = [\overleftarrow{\boldsymbol{h}}_1^{(w)}; \overrightarrow{\boldsymbol{h}}_n^{(w)}]$.

To encode the question at local level, we leverage dependency parsing to represent long-range dependencies between the answer and the focus node in p. Since the answer is denoted by the wh-word in the question, we extract the dependency path from the answer node to the focus mention in the question. Similar with Xu et al. (2016), we treat the path as the concatenation of words and dependency labels with directions. For example, the dependency path between “what” and “United States” is $\{$what, $\overrightarrow{nsubj}$, is, $\overrightarrow{prep}$, in, $\overrightarrow{pobj}$, $\langle E \rangle\}$. We apply another bidirectional GRU layer to produce the vector representation at dependency level $\boldsymbol{q}_p^{(dep)}$, capturing both syntactic features and local semantic features. Finally we combine global and local representation by element-wise addition, returning the representation of the question with respect to the semantic component, $\boldsymbol{q}_p = \boldsymbol{q}^{(tok)} + \boldsymbol{q}_p^{(dep)}$.

2.2.3 Semantic Similarity Calculation

Given the query graph with multiple semantic components, $G = \{p^{(1)}, \ldots, p^{(N)}\}$, now all its semantic components have been projected into a common vector space, representing hidden features in different aspects. We apply max pooling over the hidden vectors of semantic components, and get the compositional semantic representation of the entire query graph. Similarly, we perform max pooling for the question vectors with respect to each semantic component. Finally, we compute the semantic similarity score between the graph and question:

$$S_{sem}(q, G) = cos(\max_i \boldsymbol{p}^{(i)}, \max_i \boldsymbol{q}_p^{(i)}). \tag{1}$$

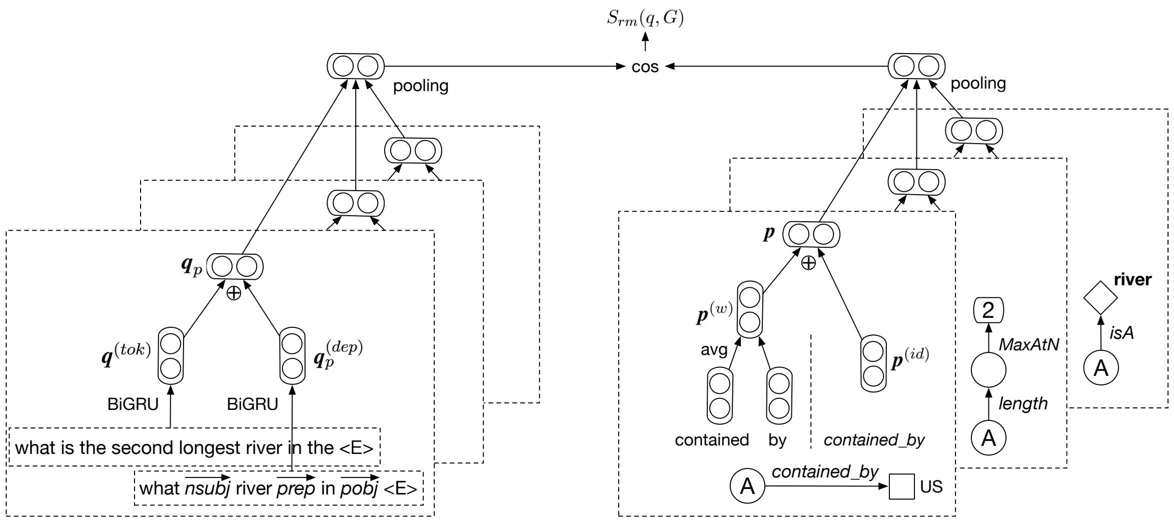

Figure 3: Overview of proposed semantic matching model.

Based on this framework, our proposed method ensures the vector spaces of the question and the entire query graph are comparable, and captures complementary semantic features from different parts of the query graph. It's worth mentioning that the semantic matching model is agnostic to the candidate generation method of the query graphs, hence it can be applied to the other existing semantic parsing frameworks.

2.3 Entity Linking Enrichment

The S-MART linker is a black box for our system, which is not extendable and tend to produce high precision but low recall linking results. To seek a better balance at entity linking, we propose an ensemble approach to enrich linking results. We first build a large lexicon by collecting all (mention, entity) pairs from article titles, anchor texts, redirects and disambiguation pages of Wikipedia. Each pair is associated with statistical features, such as linking probability, letter-tri-gram jaccard similarity and popularity of the entity in Wikipedia. For the pairs found in S-MART results, we take the above features as the input to a 2-layer linear regression model fitting their linking scores. Thus we learn a pseudo linking score for every pair in the lexicon, and for each question, we pick top-K highest pairs to enrich S-MART linking results, where K is a hyperparameter.

2.4 Training and Prediction

To predict the best query graph from candidates, we calculate the overall association score $S(q, G)$ between the question q and each candidate G, which is the weighted sum of features over entity linking, semantic matching and structural level. Table 1 lists the detail features.

During training step, we adopt hinge loss to maximize the margin between positive graphs G^+ and negative graphs G^-:

$$loss = max\{0, \lambda - S(q, G^+) + S(q, G^-)\}. \quad (2)$$

For each question, we pick a candidate graph as positive data, if the F_1 score of its answer is larger than a threshold (set to 0.1 in our work). We randomly sample 20 negative graphs G^- from the candidate set whose F_1 is lower than the corresponding G^+.

Category	Description
Entity	Sum of S-MART scores of all entities; Number of entities from S-MART; Number of entities from enriched lexicon;
Semantics	Semantic similarity score $S_{sem}(q, G)$;
Structural	Number of each kind of constraints in G; Whether a kind of constraints is used in G; Whether the main path is one-hop; Number of output answers.

Table 1: Full set of features.

3 Experiments

In this section, we introduce the QA datasets and state-of-the-art systems that we compare. We show the end-to-end results of the KBQA task, and perform detail analysis to investigate the importance of different modules used in our approach.

3.1 Experimental Setup

QA datasets: We conduct our experiments on ComplexQuestions (Bao et al., 2016), WebQuestions (Berant et al., 2013) and SimpleQuestions (Bordes et al., 2015). We use CompQ, WebQ and SimpQ as abbreviations of the above datasets, respectively. CompQ contains 2,100 complex questions collected from Bing search query log, and the dataset is split into 1,300 training and 800 testing questions. WebQ contains 5,810 questions collected from Google Suggest API, and is split into 3,778 training and 2,032 testing QA pairs. Each question is manually labeled with at least one answer entity in both datasets. SimpQ consists of more than 100K questions, and the gold answer of each question is a gold focus entity paired with a single predicate. This dataset is designed mainly for answering simple questions, and we use it for complementary evaluation.

Knowledge bases: For experiments on both CompQ and WebQ, we follow the settings of Berant et al. (2013) and Xu et al. (2016) to use the full Freebase dump [5] as the knowledge base, which contains 46M entities and 5,323 predicates. We host the knowledge base with Virtuoso engine [6]. For the experiments on SimpQ, the knowledge base we use is FB2M, which is a subset of Freebase provided with the dataset, consisting 2M entities and 10M triple facts.

Implementation detail: For all experiments in this section, we initialize word embeddings using GloVe (Pennington et al., 2014) word vectors with dimensions set to 300, and the size of BiGRU hidden layer is also set to 300. We tune the margin λ in $\{0.1, 0.2, 0.5\}$, the ensemble threshold K in $\{1, 2, 3, 5, 10, +\text{INF}\}$, and the batch size B in $\{16, 32, 64\}$. All the source codes, QA datasets, and detail results can be downloaded from `http://202.120.38.146/CompQA/`.

3.2 End-to-End Results

Now we perform KBQA experiments on WebQ and CompQ. We use the average F_1 score over all questions as our evaluation metric. The official evaluation script [7] measures the correctness of output entities at string level. While in CompQ, the annotated names of gold answer entities don't match the case of their names in Freebase, thus we follow Bao et al. (2016) to lowercase both annotated names and the output answer names before calculating the F_1 score. We set $\lambda = 0.5$, $B = 32$, $K = 3$ for WebQ and $K = 5$ for CompQ, as reaching the highest average F_1 on the validation set of each dataset.

We report the experimental results in Table 2. The result of Yih et al. (2015) on CompQ is reported by Bao et al. (2016) as their implemented result. Our approach outperforms existing approaches on CompQ dataset, and ranks 2nd on WebQ among a long list of state-of-the-art works. Jain (2016) achieves highest F_1 score on WebQ using memory networks, which is not semantic parsing based, and thus less interpretable. We point out that Xu et al. (2016) uses Wikipedia texts as the external community knowledge for verifying candidate answers, and achieves a slightly higher F_1 score (53.3) than our model, but the performance decreases to 47.0 if this step is removed. Besides, Yih et al. (2015) and Bao et al. (2016) used ClueWeb dataset for learning more accurate semantics, while based on the ablation test of Yih, the F_1 score of WebQ drops by 0.9 if ClueWeb information is removed.

Method	CompQ	WebQ
Dong et al. (2015)	-	40.8
Yao (2015)	-	44.3
Bast and Haussmann (2015)	-	49.4
Berant and Liang (2015)	-	49.7
Yih et al. (2015)	36.9	52.5
Reddy et al. (2016)	-	50.3
Xu et al. (2016) (w/o text)	-	47.0
Bao et al. (2016)	40.9	52.4
Jain (2016)	-	**55.6**
Abujabal et al. (2017)	-	51.0
Cui et al. (2017)	-	34.0
Hu et al. (2018)	-	49.6
Talmor and Berant (2018)	39.7	-
Ours (w/o linking enrich)	42.0	52.0
Ours (w/ linking enrich)	**42.8**	52.7

Table 2: Average F_1 scores on CompQ and WebQ datasets.

Our results show that entity enrichment method improves the results on both datasets by a large margin (0.8), which is a good help to our approach. We argue that the enriched results are directly comparable with other approaches, as S-MART itself is learned from semi-structured information in Wikipedia, such as anchor texts, redirect links and disambiguation pages, the enrichment step does not bring extra knowledge into our system. In addition, the improvements of the candidate generation step also show a positive effect. If we remove our implicit type filtering in Step 4 and time interval constraints in Step 5, the F_1 of CompQ slightly drops from 42.84 to 42.37. Al-

[5]detail information of the Freebase dump is available at https://github.com/syxu828/QuestionAnsweringOverFB/.

[6]http://virtuoso.openlinksw.com/

[7]The evaluation script is available at http://www-nlp.stanford.edu/software/sempre/.

though these improvements mainly concern time-related questions (around 25% in CompQ), we believe these strategies can be useful tricks in the further researches.

As a complementary evaluation, we perform semantic matching experiments on SimpQ. Given the gold entity of each question, we recognize the entity mention in the question, replace it with $\langle E \rangle$, then predict the correct predicate. Table 3 shows the experimental results. The best result is from Qu et al. (2018), which learns the semantic similarity through both attentive RNN and similarity matrix based CNN. Yu et al. (2017) proposed another approach using multi-layer BiLSTM with residual connections. Our semantic matching model performs slightly below these two systems, since answering simple questions is not the main goal of this paper. Comparing with these approaches, our semantic matching model is light-weighted, with a simpler structure and fewer parameters, thus is easier to tune and remains effective.

Method	Relation Inputs	Accuracy
BiLSTM w/ words	words	91.2
BiLSTM w/ rel_name	rel_name	88.9
Yih et al. (2015)	char-3-gram	90.0
Yin et al. (2016)	words	91.3
Yu et al. (2017)	words+rel_name	93.3
Qu et al. (2018)	words+rel_separated	**93.7**
Ours	words+path	93.1

Table 3: Accuracy on the SimpleQuestions dataset.

3.3 Ablation Study

In this section, we explore the contributions of various components in our system.

Semantic component representation: We first evaluate the results on CompQ and WebQ under different path encoding methods. Recap that the encoding result of a semantic component is the summation of its word and id path representations (Section 2.2.1), thus we compare encoding methods by multiple combinations. For encoding predicate word sequence, we use BiGRU (the same setting as encoding question word sequence) as the alternative of average word embedding. For encoding predicate id sequence, we use average predicate embedding as the alternative of the current path-level embedding ($PathEmb$).

The experimental results are shown in Table 4. The encoding method $None$ means that we don't encode the id or word sequence, and simply take the result of the other sequence as the representation of the whole component. we observe that the top three combination settings, ignoring either word or id sequence, perform worse than the bottom three settings. The comparison demonstrates that predicate word and id representation can be complementary to each other. The performance gain is not that large, mainly because predicate id features are largely covered by their word name features.

For the encoding of id sequences, $PathEmb$ works better than average embedding, consistently boosting F_1 by 0.65 on both datasets. The former method treats the whole sequence as a single unit, which is more flexible and can potentially learn diverse representations of id sequences that share the same predicates. For the encoding of word sequences, the average word embedding method outperforms BiGRU on CompQ, and the gap becomes smaller when running on WebQ. This is mainly because the training set of WebQ is about 3 times larger than that of CompQ, making it easier for training a more complex model.

Word repr.	Id repr.	CompQ F_1	WebQ F_1
None	PathEmb	41.11	51.86
Average	None	42.18	51.74
BiGRU	None	41.80	51.87
Average	Average	42.16	52.00
BiGRU	PathEmb	41.52	52.33
Average	PathEmb	**42.84**	**52.66**

Table 4: Ablation results on path representation.

Semantic composition and question representation: To demonstrate the effectiveness of semantic composition, we construct a straightforward baseline, where we remove the max pooling operation in Eq. (2), and instead calculate the semantic similarity score as the summation of individual cosine similarities: $S_{sem}(q, G) = \sum_i cos(\boldsymbol{p}^{(i)}, \boldsymbol{q}_p^{(i)})$. For methods of question encoding, we setup ablations by turning off either sentential encoding or dependency encoding.

Table 5 shows the ablation results on CompQ and WebQ. When dependency path information is augmented with sentential information, the performance boosts by 0.42 on average. Dependency paths focus on hidden features at syntactic and functional perspective, which is a good complementary to sentential encoding results. However, performances drop by 2.17 if only dependency information is used, we find that under certain dependency structures, crucial words (bolded) are not in the path between the answer and the focus mention (underlined), for example, "who did draco malloy end up **marrying**" and "who did the philippines gain **independence** from". While we observe about 5% of such questions in WebQ, it's hard to predict the correct query graph without crucial words.

In terms of semantic composition, Our max pooling based method consistently outperforms the baseline method. The improvement on WebQ is smaller than on CompQ, largely due to the fact that 85% questions in WebQ are simple questions (Bao et al., 2016). As a result of combination, our approach significantly outperforms the vanilla SP+NN approach on CompQ by 1.28, demonstrating the effectiveness of our approach. Theoretically, the pooling outcome may lead to worse end-to-end result when there are too many semantic components in one graph, because the pooling layer takes too many vectors as input, different semantic features between similar query graphs become indistinguishable. In our task, only 0.5% of candidate graphs have more than 3 semantic components, so pooling is a reasonable way to aggregate semantic components in this scenario.

Composition	Q_repr	CompQ F_1	WebQ F_1
Baseline	sentential	41.56	52.14
Baseline	both	42.35	52.39
Ours	dependency	41.48	49.69
Ours	sentential	42.59	52.28
Ours	both	**42.84**	**52.66**

Table 5: Ablation results on question representation and compositional strategy.

To further explain the advantage of semantic composition, we take the following question as an example: "who is gimli's father in the hobbit". Two query graphs are likely to be the final answer: 1) (?, *children*, *gimli_person*); 2) (?, $fictional\_children$, *gimli_character*) $\wedge$ (?, *appear_in*, *hobbit*). If observing semantic components individually, the predicate *children* is most likely to be the correct one since "'s father" is highly related and with plenty of positive training data. Both $fictional\_children$ and *appear_in* get a much lower similarity compared with *children*, hence the baseline method prefer the first query graph. In the meantime, our proposed method learns the hidden semantics of the second candidate by absorbing salient features from both predicates, and such compositional representation is closer to the semantics of the entire question than a simple "children" predicate. That's why our method manages to answer it correctly.

3.4 Error Analysis

We randomly analyzed 100 questions from CompQ where no correct answers are returned. We list the major causes of errors as follows:

Main path error (10%): This type of error occurred when the model failed to understand the main semantics when facing some difficult questions (e.g. "What native american sports heroes earning two gold medals in the 1912 Olympics");

Constraint missing (42%): These types of questions involve implicit constraints, for example, the question "Who was US president when Traicho Kostov was teenager" is difficult to answer because it implies an implicit time constraint "when Traicho Kostov was teenager";

Entity linking error (16%): This error occurs due to the highly ambiguity of mentions. For example, the question "What character did Robert Pattinson play in Harry Potter" expects the film "Harry Potter and the Goblet of Fire" as the focus, while there are 7 movies in Harry Potter series;

Miscellaneous (32%): This error class contains questions with semantic ambiguity or not reasonable. For example, the question "Where is Byron Nelson 2012" is hard to understand, because "Byron Nelson" died in 2006 and maybe this question wants to ask where did he die.

4 Related Work

Knowledge Base Question Answering(KBQA) has been a hot research top in recent years. Generally speaking, the most popular methods for KBQA can be mainly divided into two classes: information retrieval and semantic parsing.

Information retrieval based system tries to obtain target answer directly from question information and KB knowledge without explicit considering interior query structure. There are various methods (Yao and Van Durme, 2014; Bordes et al., 2015; Dong et al., 2015; Xu et al., 2016) to select candidate answers and to rank results.

Semantic parsing based approach focuses on constructing a semantic parsing tree or equivalent query structure that represents the semantic meaning of the question. In terms of logical representation of natural language questions, many methods have been tried, such as query graph (Yih et al., 2014, 2015) or RDF query language (Unger et al., 2012; Cui et al., 2017; Hu et al., 2018).

Recently, as the development of deep learning, NN-based approaches have been combined into the KBQA task (Bordes et al., 2014), showing promising result. These approaches tries to use neural network models to encode both questions and answers (or query structures) into the vector space. Subsequently, similarity functions are used to select the most appropriate query structure to generate the final answer. For example, Bordes et al. (2014) focuses on embedding the subgraph of the candidate answer; Yin et al. (2016) uses character-level CNN and word-level CNN to match different information; Yu et al. (2017) introduces the method of hierarchical residual RNN to compare questions and relation names; Qu et al. (2018) proposes the AR-SMCNN model, which uses RNN to capture semantic-level correlation and employs CNN to extract literal-level words interaction.

Belonging to NN-based semantic parsing category, our approach employs a novel encoding structure method to solve complex questions. Previous works such as Yih et al. (2015) and Bao et al. (2016) require a recognition of a main relation and regard other constraints as variables added to this main relation. Unlike their approaches, our method encodes multiple relations (paths) into a uniform query structure representation (semantic composition), which allows more flexible query structures.

There are also some works can't be simply classified in to IR based methods or SP based methods. Jain (2016) introduces Factual Memory Network, which tries to encode KB and questions in same word vector space, extract a subset of initial candidate facts, then try to employ multi-hop reasoning and refinement to find a path to answer entity. Reddy et al. (2016), Abujabal et al. (2017), and Cui et al. (2017) try to interpret question intention by templates, which learned from KB or QA corpora. Talmor and Berant (2018) attempts to answering complex questions by decomposing them into a sequence of simple questions.

5 Conclusion

To the best of our knowledge, this is the first work to handle complex KBQA task by explicitly encoding the complete semantics of a complex query graph using neural networks. We studied different methods to further improve the performance, mainly leveraging dependency parse and the ensemble method for linking enrichment. Our model becomes the state-of-the-art on ComplexQuestions dataset, and produces competitive results on other simple question based datasets. Possible future work includes supporting more complex semantics like implicit time constraints.

Acknowledgment

Kenny Q. Zhu is the contact author and was supported by NSFC grants 91646205 and 61373031. Thanks to the anonymous reviewers for their valuable feedback.

References

Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated template generation for question answering over knowledge graphs. In *WWW*, pages 1191–1200.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. *Dbpedia: A nucleus for a web of open data*. Springer.

Jun-Wei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *COLING*, pages 2503–2514.

Hannah Bast and Elmar Haussmann. 2015. More accurate question answering on freebase. In *CIKM*, pages 1431–1440.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544.

Jonathan Berant and Percy Liang. 2015. Imitation learning of agenda-based semantic parsers. *Transactions of the Association for Computational Linguistics*, 3:545–558.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250.

Antoine Bordes, Nicolas Usunier, Sumit Chopra, and Jason Weston. 2015. Large-scale simple question answering with memory networks. *arXiv preprint arXiv:1506.02075*.

Antoine Bordes, Jason Weston, and Nicolas Usunier. 2014. Open question answering with weakly supervised embedding models. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 165–180. Springer.

Qingqing Cai and Alexander Yates. 2013. Large-scale semantic parsing via schema matching and lexicon extension. In *ACL*, pages 423–433.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder-decoder approaches. *Proceedings of SSST-8*, pages 103–111.

Wanyun Cui, Yanghua Xiao, Haixun Wang, Yangqiu Song, Seung-won Hwang, and Wei Wang. 2017. Kbqa: learning question answering over qa corpora and knowledge bases. *Proceedings of the VLDB Endowment*, 10(5):565–576.

Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *ACL (1)*, pages 260–269.

Sen Hu, Lei Zou, Jeffrey Xu Yu, Haixun Wang, and Dongyan Zhao. 2018. Answering natural language questions by subgraph matching over knowledge graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(5):824–837.

Sarthak Jain. 2016. Question answering over knowledge base using factual memory networks. In *Proceedings of the NAACL Student Research Workshop*, pages 109–115.

Kangqi Luo, Xusheng Luo, and Kenny Zhu. 2015. Inferring binary relation schemas for open information extraction. In *EMNLP*, pages 555–560.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*, pages 1532–1543.

Yingqi Qu, Jie Liu, Liangyi Kang, Qinfeng Shi, and Dan Ye. 2018. Question answering over freebase via attentive rnn with similarity matrix based cnn. *arXiv preprint arXiv:1804.03317*.

Siva Reddy, Oscar Täckström, Michael Collins, Tom Kwiatkowski, Dipanjan Das, Mark Steedman, and Mirella Lapata. 2016. Transforming dependency structures to logical forms for semantic parsing. *Transactions of the Association for Computational Linguistics*, 4:127–140.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *WWW*, pages 697–706. ACM.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *NAACL-HLT*.

Christina Unger, Lorenz Bühmann, Jens Lehmann, Axel-Cyrille Ngonga Ngomo, Daniel Gerber, and Philipp Cimiano. 2012. Template-based question answering over rdf data. In *WWW*, pages 639–648. ACM.

Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question answering on freebase via relation extraction and textual evidence. In *ACL*, pages 2326–2336.

Yi Yang and Ming-Wei Chang. 2015. S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking. In *ACL-IJCNLP*, pages 504–513.

Xuchen Yao. 2015. Lean question answering over freebase from scratch. In *NAACL*, pages 66–70.

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with freebase. In *ACL*, pages 956–966.

Wen-tau Yih, Ming-Wei Chang, Xiaodong He, and Jianfeng Gao. 2015. Semantic parsing via staged query graph generation: Question answering with knowledge base. In *ACL-IJCNLP*, pages 1321–1331.

Wen-tau Yih, Xiaodong He, and Christopher Meek. 2014. Semantic parsing for single-relation question answering. In *ACL*, volume 2, pages 643–648.

Wenpeng Yin, Mo Yu, Bing Xiang, Bowen Zhou, and Hinrich Schütze. 2016. Simple question answering by attentive convolutional neural network. In *COLING*, pages 1746–1756.

Mo Yu, Wenpeng Yin, Kazi Saidul Hasan, Cicero dos Santos, Bing Xiang, and Bowen Zhou. 2017. Improved neural relation detection for knowledge base question answering. In *ACL*, pages 571–581.