Hierarchical Query Graph Generation for Complex Question Answering over Knowledge Graph

Yunqi Qiu[1,2], Kun Zhang[1,2], Yuanzhuo Wang[1,2,3], Xiaolong Jin[1,2],
Long Bai[1,2], Saiping Guan[1], Xueqi Cheng[1,2]
[1]CAS Key Laboratory of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences;
[2]School of Computer Science and Technology, University of Chinese Academy of Sciences;
[3]Big Data Academy of the Institute of Computing Technology, CAS
{qiuyunqi19b,zhangkun18z,wangyuanzhuo,jinxiaolong,bailong18b,guansaiping,cxq}@ict.ac.cn

ABSTRACT

Knowledge Graph Question Answering aims to automatically answer natural language questions via well-structured relation information between entities stored in knowledge graphs. When faced with a complex question with compositional semantics, query graph generation is a practical semantic parsing-based method. But existing works rely on heuristic rules with limited coverage, making them impractical on more complex questions. This paper proposes a Director-Actor-Critic framework to overcome these challenges. Through options over a Markov Decision Process, query graph generation is formulated as a hierarchical decision problem. The Director determines which types of triples the query graph needs, the Actor generates corresponding triples by choosing nodes and edges, and the Critic calculates the semantic similarity between the generated triples and the given questions. Moreover, to train from weak supervision, we base the framework on hierarchical Reinforcement Learning with intrinsic motivation. To accelerate the training process, we pre-train the Critic with high-reward trajectories generated by hand-crafted rules, and leverage curriculum learning to gradually increase the complexity of questions during query graph generation. Extensive experiments conducted over widely-used benchmark datasets demonstrate the effectiveness of the proposed framework.

KEYWORDS

Knowledge graph, Question answering, Semantic parsing, Query graph, Hierarchical reinforcement learning

ACM Reference Format:
Yunqi Qiu, Kun Zhang, Yuanzhuo Wang, Xiaolong Jin, Long Bai, Saiping Guan, Xueqi Cheng. 2020. Hierarchical Query Graph Generation for Complex Question Answering over Knowledge Graph. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3411888

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3411888

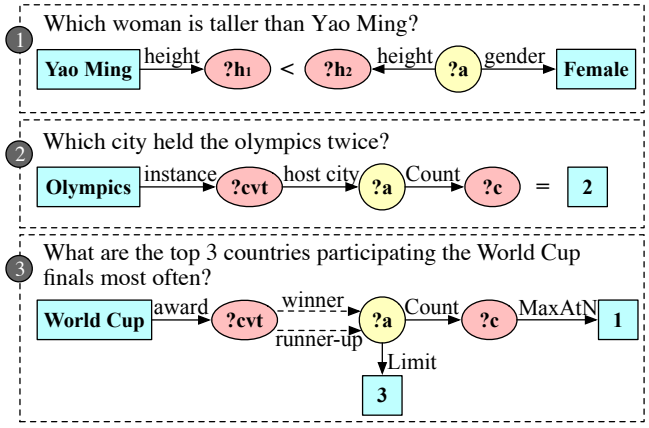


Figure 1: Example query graphs for complex questions. Constant nodes (rectangle) denote grounded KG nodes or literal information in the questions. Variable nodes (oval) and answer nodes (circle) represent ungrounded KG nodes or values. Predicate edges denote KG predicates, and functional edges indicate functional operations.

1 INTRODUCTION

With advances in information extraction, Knowledge Graphs (KGs), such as DBpedia [2], Freebase [10], and YAGO [38], are used to organize atomic facts in triple format, i.e., (subject, predicate, object). Owing to vast amounts of high-quality knowledge, these KGs have become essential resources supporting Question Answering (QA), which can benefit a variety of applications, such as web search. Generally, a KG-QA system first links the topic entity in the question to the KG, and then identifies the predicate(s) that the question refers to. Upon these two steps, the answer can be returned from the KG. A simple example is the question "How tall is Yao Ming?", which relies on the triple (*Yao Ming*, *height*, *2.26m*).

Although a simple question can be answered by a single triple, many complex questions require more information and even functional operations, such as comparison, aggregation, and sorting. To understand the compositional semantics of complex questions, one practical method is learning a Semantic Parser (SP), which converts an unstructured question into its structured representation. Traditional approaches [1, 5, 8, 50] rely on templates to do so, which inevitably have limited coverage. Recent methods leverage *query*

graphs to represent the semantic structures of questions, as shown in Figure 1. In [18, 19, 33], questions are first converted to semantic graphs via a parser, e.g., Combinatory Categorial Grammar (CCG) parser and dependency parser. These semantic graphs are mapped to KG subgraphs, and then transformed into query graphs according to rules. Free from existing parsers, in [4, 9, 23, 25, 47, 49], a query graph is formed by directly connecting the topic entity with the answer node through a relation path in the KG. Then constraints are attached to the relation path according to heuristic rules. However, there are still two challenges to be addressed.

Challenge 1. Complex semantic structures. Some questions have complex semantic structures that cannot be expressed by the query graphs generated by existing methods. Specifically, there are three types of questions not well studied in existing works: (1) *Unconnected answer nodes.* In some cases, answer nodes are not in the topic-entity-centric subgraphs in the KG, and they are connected by functional operations in the query graphs, such as the first example in Figure 1. But existing methods cannot work for such types of complex questions. (2) *Varied aggregation functions.* As shown in the second and third examples in Figure 1, some questions need aggregation functions like *Count*, *Limit*, etc. But existing methods consider only *Count*, and treat it as a node instead of an edge, making it impractical to conduct further operations (e.g., comparison and sorting) over the aggregation value. Moreover, they recognize aggregation functions by a predefined word list, but the two examples have no trigger words they defined. (3) *Logical disjunction semantics.* Existing methods combine all the triples in a query graph with logical conjunction, while some questions express alternative propositions combined with the coordinator "or". In these cases, the corresponding triples should be combined with logical disjunction, e.g., (*?cvt*, *winner*, *?a*) and (*?cvt*, *runner-up*, *?a*)[1] both indicate that someone has reached the final in the third example.

Challenge 2. Training from weak supervision. Considering the cost of fine-grained annotations, it is impractical to exactly label the gold query graph for each question, and only the final answer is labeled, as weak supervision. Existing methods rely on heuristic rules or existing parsers to determine the structure of query graphs, making their models cannot be trained in an end-to-end manner and suffer from error cascading.

To overcome the aforesaid challenges, we propose a Director-Actor-Critic framework, and utilize options over a Markov Decision Process (MDP) [40, 41] to formulate query graph generation as a hierarchical decision problem. In our formulation, triples in query graphs fall in different categories to represent different semantic information of the questions. At a specific time step, the Director selects a high-level action (option) to determine which category of triple needs to be generated. Then, the Actor chooses a series of low-level actions (primitive actions) to generate the corresponding triple stepwise, and the Critic calculates a semantic score between the newly generated triple and the given question. Within this framework, there is no need to limit the answer nodes to the topic-entity-centric subgraphs. Triples, functional operations, and logical connection types are generated by a trainable model instead of heuristic rules.

[1] A compound value type (CVT) node is a kind of auxiliary nodes in Resource Description Framework (RDF) to maintain N-ary facts.

To train from weak supervision, we base the proposed framework on hierarchical Reinforcement Learning (RL) with intrinsic motivation [21, 37]. Specifically, the Director is rewarded by extrinsic reward signals provided by the outside environment, and the Actor is rewarded by intrinsic motivation computed by the Critic. With this strategy, our model can be trained in an end-to-end manner. Compared with typical flat RL methods such as DQN [28] and the REINFORCE algorithm [43], our strategy helps alleviate the credit assignment problem arising from long decision trajectories. Meanwhile, samples from a randomly initialized policy often attain small rewards, resulting in a slow training process in the initial phase. To make search more efficient in a sizeable state-action space, we pre-train the Critic with high-reward query graphs generated by hand-crafted rules in previous works, inspired by AlphaGo [36]. Based on these high-reward trajectories, we leverage curriculum learning [6] to train the Director and the Actor gradually according to the complexity of questions during query graph generation.

In summary, we make the following contributions in this paper:

- We propose a new framework and formulate query graph generation as a hierarchical decision problem, overcoming the challenge of complex semantic structures;
- We leverage hierarchical RL with intrinsic motivation to train from weak supervision, and utilize pre-training and curriculum learning to accelerate the training process;
- We demonstrate the effectiveness of the proposed framework through extensive experiments and careful ablation studies on widely-used benchmark datasets.

2 METHOD

In this section, we first introduce the definition of the query graph (Sec. 2.1), and formulate the query graph generation as a hierarchical decision process (Sec. 2.2). Then we describe constant node linking, which is the foundation of the query graph generation (Sec. 2.3). Next, we introduce the question encoder (Sec. 2.4) and the graph encoder (Sec. 2.5), both of which provide constituents of the states in the decision process. We show the algorithm of training the agent with hierarchical RL with intrinsic motivation in Sec. 2.6.

2.1 Query Graph

As described in [47], a query graph is a restricted subset of λ-calculus in the graph representation. Thus, it can be straightforwardly translated to an executable query language, e.g., SPARQL [31]. For covering more types of complex questions, we tune the original definition to improve the expressivity. As shown in Figure 1, our query graph consists of three types of nodes: constant node (rectangle), variable node (oval), answer node (circle). A constant node can be a grounded KG entity or type, such as *Yao Ming* and *Female*, and it can also represent literal information in a given question, e.g., the values *1* and *3* in the third example. Variable nodes and answer nodes represent ungrounded KG nodes or values. There are two types of edges: predicate edge and functional edge. A predicate edge represents a KG predicate, such as *height*. A functional edge indicates a functional operation, such as *<*, *MaxAtN*, and *Limit*.

Considering the nodes and edges, we classify triples in query graphs into five types: "Basic", "Union", "Filter", "Ordinal", and "Aggregation". Specifically, in "Basic" and "Union", edges are KG

predicates, subjects and objects are grounded or ungrounded KG nodes, while in "Union", the two nodes are connected by more than one edge, and the corresponding triples are combined with logical disjunction. The edges in "Filter" indicate numerical or temporal comparison, including $<, \leq, >, \geq, =, \neq$. In "Ordinal", the subject needs to be sorted, the edge represents whether the ranking order is ascending (i.e., *MinAtN*) or descending (i.e., *MaxAtN*), and the object indicates the ordinal number. In "Aggregation", edges indicate aggregation functions over subjects, including *Count*, *Limit*, etc, and the objects are aggregation results or limit over quantity.

2.2 Director-Actor-Critic

The underlying formulation frequently used in RL is an MDP [40], which is meant to be a straightforward framing of the problem of learning from the interaction between the agent and the environment. The agent is the learner and decision-maker, and the environment comprises everything outside the agent. More specifically, a traditional MDP is defined as a tuple (S, A, p, R, γ), where S is the state space, A is the action space, $p : S \times A \times S \to [0, 1]$ denotes the state-transition probabilities, $R : S \times A \times S \to \mathbb{R}$ denotes the extrinsic reward function, and γ is the discounted factor. At each time step t, the agent observes the environment's state, $S_t \in S$, and selects an action $A_t \in A(S_t)$. As a consequence of its action, the environment changes to a new state S_{t+1} with probability $p(S_{t+1}|S_t, A_t)$, and the agent receives a numerical reward based on the extrinsic reward function $R_{t+1} = R(S_t, A_t, S_{t+1})$. If framed by a traditional MDP, a query graph can be constructed by joining one constituent (a node or an edge) at a time. Such a computation plan of this granularity for a complex question will be error-prone and result in the long-term credit assignment problem.

Considering the triple types, we formulate query graph generation by options over an MDP [41], where each option represents generating one corresponding type of triple. Different from the traditional MDP setting where an agent can only choose a primitive action at each single time step, with options, the agent can choose a "multi-step" action, which is the temporal abstraction of primitive actions. As is illustrated in Figure 2, the proposed hierarchical query graph generation framework, namely Director-Actor-Critic (DAC), integrates three modules:

1. **Director** selects an option o_t to determine which type of triple is needed according to the current state s_t.
2. **Actor** selects an action a_t using s_t and o_t, and the option o_t remains for the next few time steps until it is achieved.
3. **Critic** is responsible for evaluating whether o_t has been reached by calculating the semantic similarity between the newly generated triple and the question.

As is proved by [41], for any MDP, and any set of options defined on that MDP, the decision process that selects only among those options, executing each to termination, is a Semi-MDP (SMDP). The main components of the SMDP are presented as follows.

States. At time step t, state $S_t = (q, h_t) \in S$ is a tuple where q is the given question, and h_t is the decision history at time step t. Note that for the Director, h_t is just the current query graph g_t, while for the Actor, it also involves the primitive actions selected in the current option besides g_t.

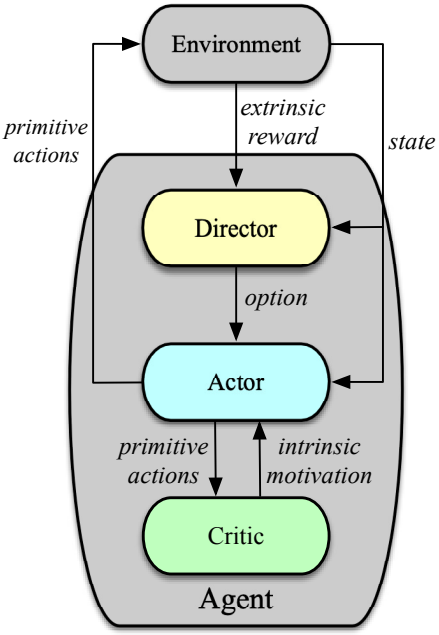


Figure 2: Overview of the Director-Actor-Critic framework based on options over an MDP.

Options. Following [41], an option consists of three components: an initiation set $I \in S$ containing states where the option can be initiated, an intra-option policy $\pi : S \times A \to [0, 1]$ that selects primitive actions for the option, and a termination condition $\beta : S \to [0, 1]$ that specifies when the option is completed. An option $\langle I, \pi, \beta \rangle$ is available in state s_t if and only if $s_t \in I$. If the option is taken, then primitive actions are selected according to π until the option terminates according to β. For the options "Basic" and "Union", the initiation set I_p equals to the whole set S. While for the options "Filter", "Ordinal", and "Aggregation", there must be variable or answer nodes in the query graph. Each of these five options terminates after a corresponding triple is generated.

Primitive Actions. For each option, there are three types of primitive actions: selecting an existing node in the current query graph, adding a predicate or functional edge according to the chosen option, and connecting a newly generated node or another existing node to the first selected node with the generated edge. Meanwhile, each option has its own list of valid primitive actions at each step. For the options "Basic" and "Union", only the grounded or ungrounded entities and types can be selected at first, and then all the KG predicates linked to the selected node are taken into consideration. For the options "Filter" and "Ordinal", only variable and answer nodes that represent temporal or numerical values can be first selected, and the chosen object should be a constant literal node. For the option "Aggregation", ungrounded entities or CVT nodes can be selected at the first step.

Transition. Due to the setting of primitive actions and states, the probability values of state transition are deterministic.

Extrinsic Rewards. Extrinsic rewards are special signals passed from the environment to the agent, and they are formulations of the

agent's purpose. Since only the final answers are labeled as weak supervision, in our formulation, F_1 scores of predicted answers are treated as the extrinsic rewards, and only options instead of primitive actions will be rewarded.

Intrinsic Motivation In contrast to extrinsic rewards, intrinsic motivation is an internal reward not from the environment. It is a kind of additional reward to the agent to help detect how the option is completed after a series of primitive actions. To evaluate whether the option has been reached, we measure the semantic similarity between the given question and the generated triple. Details will be introduced in Sec. 2.6. With the combination of extrinsic rewards and intrinsic motivation, long-term credit assignment problem will be alleviated.

2.3 Constant Node Linking

A constant node represents a grounded KG entity or type, and it can also represent temporal or numerical information in a given question. Given a natural language question, we should first recognize these constant nodes.

For entity linking, we generate (mention, entity) pairs with the help of S-MART [45], which is the state-of-the-art entity linking tool. For type linking, we first label each word in the question with its part-of-speech (POS) tag via Stanford CoreNLP toolkit [26], and extract all the nouns and proper nouns. Then words contain synonyms of a KG type are picked out to form (mention, type) pairs. For time linking and numeral linking, we label the question with the Named Entity Recognition (NER) annotator of CoreNLP, and get candidate temporal nodes or numerical nodes. Note that if the question contains a superlative phrase with no word labeled as the ordinal number, the integer one is added as the ordinal number.

2.4 Question Encoder

A GRU network [14] is adopted to convert the natural language question q into vectors. Note that if we use unidirectional GRU, the output state of a specific word contains only the information of the words before it in the question, while the words after it are ignored. Thus, we employ bidirectional GRU, which consists of both forward and backward networks, and the distributed representation of the question can be obtained by averaging all the output states.

Formally, for question $q = (w_1, w_2, \cdots, w_n)$, where n denotes the word number, each word w_i $(i = 1, 2, \cdots, n)$ is initialized with a dense word embedding. Afterwards, we feed the sequence of word embeddings into a bidirectional GRU network, and obtain a series of d-dimensional output states $\boldsymbol{w_1}, \boldsymbol{w_2}, \cdots, \boldsymbol{w_n}$. Here $\boldsymbol{w_i} = [\overrightarrow{\boldsymbol{cw_i}}; \overleftarrow{\boldsymbol{cw_i}}]$, where $\overrightarrow{\boldsymbol{cw_i}}$ is the $d/2$-dimensional output state for word w_i generated by the forward GRU, and $\overleftarrow{\boldsymbol{cw_i}}$ is from the backward GRU with the same dimension. Then the question representation is calculated by

$$\boldsymbol{q} = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{w_i} \in \mathbb{R}^d. \tag{1}$$

2.5 Graph Encoder

Considering that query graphs are dynamically updated, it is rather costly to encode them using graph neural networks [35, 46] due to calculating the adjacency matrices every time. As is illustrated in Figure 3, our graph encoder is comprised of two modules: an LSTM network which encodes triples in the query graph separately, and a transformer model which captures the interrelations between these triples and generates the representation of the whole graph without adjacency matrices.

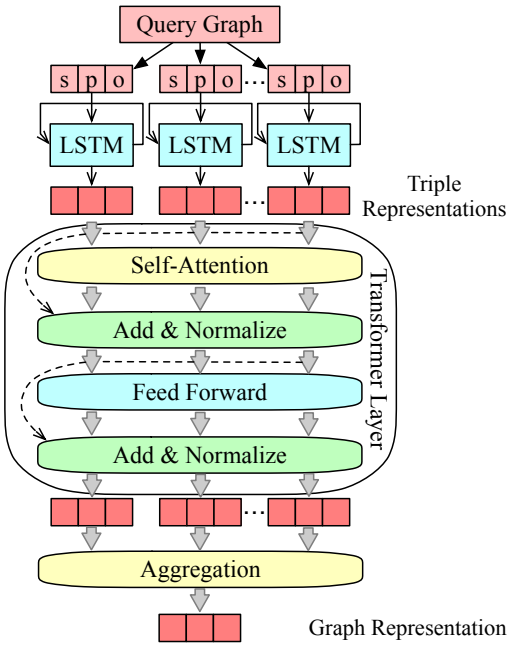


Figure 3: Overview of the architecture of the graph encoder comprised of two modules: (1) an LSTM network which encodes each triple in the query graph; (2) a transformer which captures the global correlation between triples.

As mentioned above, a query graph is comprised of triples, each of which reflects part of the semantic information of the given question. To obtain the graph representation, we first encode all the triples. Specifically, we feed the sequence of subject, predicate, and object into an LSTM network [17] with the hidden dimension $d/3$. Then, the output states at the three steps are concatenated as the triple representation $\boldsymbol{tr} \in \mathbb{R}^d$. Note that these triple constituents are initialized as follows: for a constant node, we tokenize its mention extracted in Sec. 2.3, and each token is initialized with a dense word embedding. After feeding the word embeddings into a GRU network, we generate a distributed representation for the constant node by averaging the GRU output states. KG predicates are initialized similarly, and each functional predicate is assigned with an embedding. For a newly generated variable node or answer node, we feed the word embedding of "variable" or "answer" into the LSTM network, and the final output state of the triple sequence is treated as its embedding.

The LSTM network encodes different triples separately, ignoring the interrelations between them, and these relationships are not explicitly indicated, such as dependency arcs. To capture the compositional semantics from a global perspective, we employ a transformer [42] which attends to different parts of a graph efficiently. Based on self-attention mechanism, the adopted encoder

weights the importance between these triples, and updates each representation with all the triples in the query graph.

Formally, to learn the representation of the whole graph with m triples, all the triple representations generated by the LSTM are firstly stacked as $Q, K, V \in \mathbb{R}^{m \times d}$ ($Q = K = V$). Then soft selection is applied with the scaled dot-product attention. Before and after fed into a fully connected feed-forward network, residual connection [16] and layer normalization [3] are employed. Finally, the graph representation $\boldsymbol{g} \in \mathbb{R}^d$ is generated through max pooling over the column vectors of the output as follows:

$$\boldsymbol{g} = MaxPooling(\{Z_2\}_{col}), \tag{2}$$

$$Z_2 = LayerNorm(Z_1 + Z_{ffu}), \tag{3}$$

$$Z_{ffu} = ReLU(Z_1 \cdot W_1 + b_1) \cdot W_2 + b_2, \tag{4}$$

$$Z_1 = LayerNorm(Q + Z), \tag{5}$$

$$Z = \sigma(\frac{Q \cdot K^T}{\sqrt{d}})V, \tag{6}$$

where σ is the SoftMax operator, $W_1, W_2 \in \mathbb{R}^{d \times d}$ and $b_1, b_2 \in \mathbb{R}^{m \times d}$ are all learned parameter matrices.

2.6 Hierarchical Decision and Optimization

The search policies for the Director and the Actor are parameterized using the state information and the decision history. At time step t, the Director receives state s_t, and selects an option according to the calculated probability distribution. Besides the five options, a "Loop" option is added, which makes no change to the query graph. Each option is assigned with a dense embedding with dimension d. The option space is encoded by stacking the embeddings of all valid options in O_t: $\boldsymbol{O_t} \in \mathbb{R}^{|O_t| \times d}$. And the policy network for the Director π_D is defined as

$$\pi_D(o_t|s_t) = \sigma(\boldsymbol{O_t} \cdot W_{D_2} \cdot LeakyReLU(W_{D_1} \cdot [\boldsymbol{g_t}; \boldsymbol{q}])), \tag{7}$$

where LeakyReLU [24] is an activation function, $W_{D_1} \in \mathbb{R}^{d \times 2d}$ and $W_{D_2} \in \mathbb{R}^{d \times d}$ are learned parameter matrices, $\boldsymbol{g_t}$ is the representation of the query graph at time step t, and $\boldsymbol{q}$ is the question representation.

Once an option o_t is selected by the Director, the Actor selects a series of primitive actions to complete the option. As mentioned above, the decision history consists of the current query graph and the primitive actions selected during o_t: $\boldsymbol{h_{t+i}} = [\boldsymbol{g_t}; \boldsymbol{h_{o_t,i}}], i = 0, 1, 2$, and we sum the embeddings of selected primitive actions as $\boldsymbol{h_{o_t,i}} \in \mathbb{R}^d$. Note that $\boldsymbol{h_{o_t,0}}$ is a zero vector. The action space is encoded by stacking the embeddings of all valid actions in A_{t+i}: $\boldsymbol{A_{t+i}} \in \mathbb{R}^{|A_{t+i}| \times d}$. And the policy network for the Actor π_A is defined as

$$\pi_A(a_{t+i}|s_{t+i}, o_t) = \sigma(\boldsymbol{A_{t+i}} \cdot W_{A_2} \cdot LeakyReLU(W_{A_1} \cdot [\boldsymbol{h_{t+i}}; \boldsymbol{q}; \boldsymbol{o_t}])), \tag{8}$$

where $W_{A_1} \in \mathbb{R}^{d \times 4d}$ and $W_{A_2} \in \mathbb{R}^{d \times d}$ are learned parameter matrices. When a new triple is generated and added to the query graph, the option o_t terminates and the Director selects a new option. When the decision process is completed, the answer can be obtained through the generated query graph.

The optimization is done with a classical policy gradient method, i.e., the REINFORCE algorithm [43], which uses the current policy to roll out multiple trajectories to estimate a stochastic gradient, and then updates the policy through stochastic gradient ascent.

Algorithm 1 Training Director-Actor-Critic for KG-QA

Input:
Training dataset $D = \{(q, a)\}$; Total training iterations N; Agent Training epochs N_a

```
Find good query graphs G+ for some questions in D
for iteration = 1 to N do
  for (q, g+) ∈ G+ do
    Sample negative examples tr− for each triple tr+ in g+
    Calculate the loss by Eq. 16
  end for
  Optimize the Critic
  for epoch = 1 to N_a do
    for (q, a) ∈ D do
      Calculate the question representation by Eq. 1
      while True do
        Calculate the graph representation by Eq. 2
        Sample an option by Eq. 7
        for t = 1; t ≤ 3; t++ do
          Sample an action by Eq. 8
        end for
        Calculate the triple representation by the LSTM
        Calculate the intrinsic motivation by Eq. 10
      end while
      Obtain the extrinsic rewards
    end for
    Optimize the Director by Eq. 14
    Optimize the Actor by Eq. 15
  end for
  Add better query graphs to G+
end for
```

To optimize the high-level policy, we maximize the expected cumulative extrinsic rewards over all the question-answer pairs. For a query graph g, the F_1 score of its predicted answers is treated as its terminal reward $F_1(q, g)$. Thus, the cumulative extrinsic rewards for the option o_t is $G_t = \sum_{k=t}^{T} \eta^{k-t} F_1(q, g)$, where $\eta \in (0, 1)$ is the discount factor. Then, the object function for the high-level policy is computed as follows:

$$J(\theta_D) = \mathbb{E}_{(q,a) \in D}[\mathbb{E}_{o_1, o_2, \cdots, o_T \sim \pi_D}[\sum_{t=1}^{T} \eta^{t-1} G_t]]. \tag{9}$$

Similarly, the low-level policy π_A is optimized by maximizing the expected cumulative intrinsic motivation from the Critic. Specifically, during option o_t, the actor samples along π_A and adds a triple to the query graph. The triple representation $\boldsymbol{tr} \in \mathbb{R}^d$ is obtained as mentioned in Sec. 2.5, and the Critic calculates the semantic similarity between the triple and the question. For the purpose of making the Critic know which part of the question should be focused on at present, we firstly cast $\boldsymbol{tr}$ onto the embedding space of q to measure the similarity between tr and $\boldsymbol{w_i}$ ($i = 1, 2, \cdots, n$). We pass the results through a SoftMax operator, yielding an attention distribution over the question word embeddings. Then, a weighted sum of these vectors $\boldsymbol{q_*} \in \mathbb{R}^d$ is produced as the result

of the interaction between the triple and the question according to the attention weights. Finally, the intrinsic motivation for the Actor is computed by a multi-layer perceptron as follows:

$$r_c = Sigmoid(W_{C_2} \cdot LeakyReLU(W_{C_1} \cdot [\boldsymbol{tr}; \boldsymbol{q_*}])), \tag{10}$$

$$\boldsymbol{q_*} = \sum_{i=1}^{n} attn_i \cdot \boldsymbol{w_i}, \tag{11}$$

$$attn_i = \sigma(W_3 \cdot (\boldsymbol{tr} \odot \boldsymbol{w_i}) + b_3), \tag{12}$$

where $W_{C_1} \in \mathbb{R}^{d \times 2d}$, $W_{C_2} \in \mathbb{R}^{d \times d}$, $W_3 \in \mathbb{R}^d$, and $b_3 \in \mathbb{R}$ are learned parameters, and $\odot$ represents Hadamard multiplication. Thus, the object function for the low-level policy is computed as follows:

$$J(\theta_A) = \mathbb{E}_{q, \{o_j\}_{j=1}^{T}}[\mathbb{E}_{a_1, a_2, a_3 \sim \pi_A}[\sum_{t=1}^{3} \eta^{t-1} R_t]], \tag{13}$$

where $R_t = \sum_{k=t}^{3} \eta^{k-t} r_c$ denotes the cumulative intrinsic motivation for the action a_t. With the likelihood ratio trick, the gradients for the high-level policy and low-level policy yield:

$$\nabla_{\theta_D} J(\theta_D) = \mathbb{E}_{(q,a) \in D}[\mathbb{E}_{o_1, o_2, \cdots, o_T \sim \pi_D}[\sum_{t=1}^{T} \eta^{t-1} G_t \nabla_{\theta_D} \log \pi_D]], \tag{14}$$

$$\nabla_{\theta_A} J(\theta_A) = \mathbb{E}_{q, \{o_j\}_{j=1}^{T}}[\mathbb{E}_{a_1, a_2, a_3 \sim \pi_A}[\sum_{t=1}^{3} \eta^{t-1} R_t \nabla_{\theta_A} \log \pi_A]]. \tag{15}$$

Note that if randomly initialized, the Critic will not give a correct evaluation for the triples. Inspired by AlphaGo, we pre-train the Critic with high-reward trajectories generated by hand-crafted rules in [4, 47]. Specifically, we first collect relation paths that achieve $F_1 \geq 0.5$. If a path-only query graph returns more entities than the gold answers, we explore adding constraint nodes, until the entities retrieved by the query graph are identical to the gold answers, or the F_1 score cannot get increased more. Thus, we can form the parallel question and good (not gold) query graph pairs. For each triple in a good query graph, we sample negative examples, mainly including other KG predicates and different functional edges. During the pre-training step, we adopt hinge loss to maximize the margin between good triples tr^+ and negative triples tr^- as follows:

$$loss_c = Max\{0, \gamma - r_c(tr^+) + r_c(tr^-)\}, \tag{16}$$

where γ is the margin. Every several epochs, query graphs generated by the model that achieve higher F_1 scores are added to train a better Critic, which is similar to multi-task learning. Algorithm 1 describes our training procedure for the proposed Director-Actor-Critic framework.

Meanwhile, it is not easy to generate proper query graphs for some complex questions with a randomly initialized policy. Because of the sizeable search space, random samples often attain small rewards, resulting in a slow training process in the initial phase. To solve this problem, we leverage curriculum learning [6] to control the search space. Based on the good query graphs generated by hand-crafted rules, these complex questions are classified according to both the types and number of options used in the query graph generation. Then we apply curriculum learning by gradually increasing both these quantities when training the high-level and low-level policies. Specifically, we first train the agent on simple questions constrained to only use the option "Basic", and the maximum number of options is 1. Then, the maximum number of options gets increased, while no other type of options is involved. Finally, the agent is trained on more complex questions requiring other types of options.

3 RELATED WORK

KG-QA: Up to now, there are two mainstream branches for KG-QA. One branch learns a Semantic Parser, which converts an unstructured question into its structured representation. Traditional SP-based approaches [1, 5, 8, 50] rely on templates. Recent works [22, 34] employ high-level programming languages, and treat SP as neural program induction. These methods demand users to be familiar with both the syntax and the back-end data structures. Meanwhile, mismatches between syntax predicted structures and KG structures limit the performance. To bridge this gap, query graphs are leveraged to represent semantic structures of questions. In [18, 19, 33], natural language questions are first converted to semantic graphs via an existing parser. Then these semantic graphs are mapped to the KG, and the query graph generation is conceptualized as a graph matching problem. STAGG [47] chooses one grounded entity as the topic entity, and identifies the relation path between the topic entity and the answer. To restrict the search space, it only explores all paths of length 2 in the topic-entity-centric subgraph when the middle variable is a CVT node, and paths of length 1 otherwise. Then, it attaches constraints to the paths according to heuristic rules. Finally, all the candidate query graphs are ranked by a scoring function. Following [47], more constraints are considered in [4], and the scoring function is specially designed to better encode the query graphs in [23, 25, 49]. In [9], the original query graph generation is repeated several times to cover questions with longer relation paths and more constraints.

Besides SP-based one, the other branch (e.g., [11, 13]) retrieves candidate answers from the topic-entity-centric subgraph, and encodes the question and each candidate as semantic vectors in a common embedding space. The rank of these candidates is based on semantic similarities between their vectors. With the progress of deep learning, neural networks receive tremendous attention and are employed to generate better distributed representations of questions and KG constituents [12, 15]. Some works (e.g., [39, 44]) incorporate external knowledge such as Wikipedia free text to verify candidate answers. Some works (e.g., [27, 32]) focus on enhancing the capability of multi-hop reasoning. These methods can be trained in an end-to-end manner, but fail to answer complex questions that require functional operations. As is known, neural networks achieve great success with sensory perception, but they are weak in learning non-differentiable operations.

Intrinsic motivation: The nature and origin of intrinsic reward functions is a standing question in RL. Agents with intrinsic reward structures are explored in [37] in order to learn generic options that can apply to a wide variety of tasks. In [21], there is a minimal assumption of a binary intrinsic reward i.e. 1 if the goal of the selected option is reached and 0 otherwise. This setting is practical in some tasks such as the ATARI game "Montezuma's Revenge" [21], and

task-completion dialogue [29]. However, considering the complex semantic information of natural language questions, this setting is impractical in query graph generation.

4 EXPERIMENT

To evaluate the proposed Director-Actor-Critic (DAC) framework over complex questions, we conduct experiments on three widely-used benchmark datasets: WebQuestions and two different datasets with the same name, i.e., ComplexQuestions (Sec. 4.1). Model implementation details are introduced in Sec. 4.2. Experimental results and ablation studies demonstrate the effectiveness of DAC (Sec. 4.3 and Sec. 4.4), and error analysis lists the major causes of answering questions incorrectly (Sec. 4.5).

4.1 Datasets

The three adopted datasets are all from the general domain and based on Freebase.

Specifically, WebQuestions [7] (WebQ) consists of 5,810 question-answer pairs with 3,778 questions (65%) for training and 2,032 questions (35%) for testing respectively. These questions are collected by Google Suggest API and the answers are fetched from Freebase with the help of Amazon Mechanical Turk. This dataset is widely-used in recent papers on semantic parsing and KG-QA. The most important and unique property that makes it appealing is that the distribution of this question set is closer to the "real" information need of search users than that of a small group of human editors. Questions generated from templates are not varied, and usually make models trapped in overfitting. Although the questions in WebQuestions are not directly sampled from search query logs, the selection process is still biased to commonly asked questions on a search engine, making the questions tend to be more natural.

The first ComplexQuestions [4] (CompQ-1) consists of 2,100 question-answer pairs from three sources: WebQuestions, ComplexQuestions[2] [48], and manually labeled QA pairs by [4]. The dataset is split into training and test sets, which contain 1,300 questions (62%) and 800 questions (38%), respectively. Compared with WebQuestions, these questions require more complex constraints.

The second ComplexQuestions [1] (CompQ-2) is constructed with the crawl of WikiAnswers[3] and human annotation. The dataset is composed of 150 test questions and contains no training questions. All the questions exhibit compositional semantics through multiple clauses, e.g., "Which US presidents were Duke university graduates?".

Since Freebase has been shut down, we use the full Freebase dump provided by [44] for experiments on WebQ, CompQ-1, and CompQ-2. The dump contains 46M entities and 5,323 predicates, and we host it with the Virtuoso engine[4].

4.2 Implementation Details

We initialize the word embeddings with pre-trained ones introduced by [30]. The embedding size is $d = 300$. For the bidirectional GRU as question encoder, we set the number of layers to 2 and its hidden dimension to 150. The unidirectional GRU as the KG constituent encoder has 2 layers of hidden dimension 300, and the LSTM in the graph encode has 2 layers of hidden dimension 100. Meanwhile, we apply dropout to all the neural networks with the dropout rate $= 0.3$. For REINFORCE algorithm, we tune the discount factor η within $(0.9, 1.0)$, and for the Critic, we set the margin $\gamma = 0.5$. We adopt the ADAM optimizer [20] for parameter optimization with an initial learning rate $lr = 0.001$.

4.3 Results and Discussion

Following [47], we measure the model performance basically by the ratio of correctly answered questions. Because some questions have more than one answer, we compute precision, recall and F_1 based on the model output for each individual question, and the average F_1 score is reported as the main evaluation metric.

Table 1: Average F_1 scores (%) on benchmark datasets.

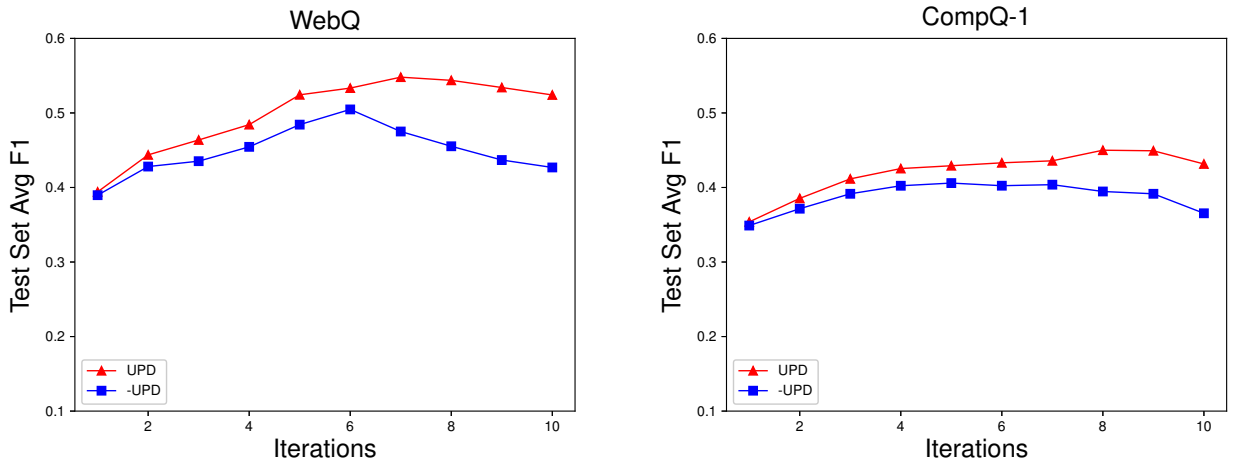
Method	WebQ	CompQ-1	CompQ-2
MulCNN [15] (2015)	40.8	-	-
QAoverFB [44] (2016)	47.0	-	-
Aqqu [5] (2015)	49.4	-	27.8
QUINT [1] (2017)	51.0	-	49.2
GraphParser [33] (2014)	41.3	-	-
NFF [18] (2018)	49.6	-	-
STF [19] (2018)	53.6	-	54.3
STAGG [47] (2015)	52.5	36.9	-
MulCG [4] (2016)	52.4	40.9	-
CompQA [23] (2018)	52.7	42.8	-
Tree2Seq [49] (2020)	52.1	-	-
DAC	**54.8**	**45.0**	**59.3**

Table 1 shows the average F_1 scores on WebQ, CompQ-1 and CompQ-2. We compare our approach with both embedding-based and SP-based methods, and all the methods are trained from weak supervision. Since there are no training data in CompQ-2, we utilize the model trained on WebQ directly following [1, 19]. As is reported, DAC outperforms all the baselines on the three benchmark datasets.

MulCNN [15] and QAoverFB [44] formulate question answering as a retrieval problem in the topic-entity-centric subgraph following [11]. MulCNN introduces multi-column convolutional neural networks to encode questions from three different aspects, namely, answer path, answer context, and answer type. QAoverFB encodes both syntactic and sentential information through multi-channel convolutional neural networks. When using Wikipedia texts as the external knowledge to verify candidate answers, it can achieve $F_1 = 53.3$ on WebQ. But the task formulation of the two methods limits the model performance on complex questions, and cannot handle those requiring non-differentiable functional operations.

Aqqu [5] and QUINT [1] are both template-based approaches. Aqqu designs three query templates for WebQ and tries to match questions to these templates. If applied on CompQ-2 directly, it can only achieve $F_1 = 27.8$. In [19], questions are manually decomposed to subquestions first. Then Aqqu takes these subquestions as input, and the predicted answers are the intersection of subquestions'

[2]This dataset is also called ComplexQuestions, focusing on open domain complex questions with prepositional/adverbial constraints.
[3]http://wiki.answers.com
[4]https://github.com/openlink/virtuoso-opensource



(a) (b)

Figure 4: Illustration of performance on test sets. The red line shows the metric changes of DAC on test sets, and the blue line belongs to the ablated model without updating the Critic.

answer sets. In this situation, Aqqu achieves better performance (F_1 = 46.7). Compared with Aqqu, QUINT leverages a dependency parser and automatically learns utterance-query templates solely from user questions paired with their answers. However, templates have limited coverage, resulting from the variability of natural language and the large scale of the KG.

GraphParser [33] formulates query graph generation as a graph matching problem. It converts the output of the CCG parser into a graphical representation and subsequently maps it onto the KG subgraphs by replacing edges and nodes with relations and types in the KG. However, there still exist structural mismatches between the output of the CCG parser and the KG, limiting the model performance. NFF [18] and STF [19] propose several primitive operations to bridge the structural gap. However, these methods rely on sophisticated parsers and may suffer from error cascading once obtaining an incorrect parse. When faced with complex questions with multiple clauses in CompQ-2, DAC exceeds STF by a larger margin due to a more flexible framework.

We notice that STAGG [47] performs not very well on CompQ-1, where questions require more constraints than WebQ. MulCG [4] and CompQA [23] take account of these constraints and achieve higher F_1 scores than STAGG. However, during query graph generation, these methods consider the whole topic-entity-centric subgraph to determine a relation path, and the disturbance caused by other knowledge items within 2 hops of the topic entity may mislead the training and testing processes, especially in a large scale KG. In contrast, DAC formulates the task as a hierarchical decision problem, at each high-level step, the options "Basic" and "Union" only add one KG triple to the query graph. Thus, there is no need to consider the whole topic-entity-centric subgraph. Moreover, each type of option has its own valid primitive action space, and the search space gets restricted significantly. Meanwhile, the previous methods rely on heuristic rules to add constraints, limiting its coverage. In our formulation, complex constraints are generated by the model, and mistakes will be fixed during training, which helps get more accurate answers.

Table 2: Performance on testing questions with different complexity on WebQ.

#Options	1	2	3	4	≥ 5
Percentage (%)	56.8	25.4	12.1	1.5	4.2
F_1 (%)	56.0	52.3	54.9	53.4	54.2

As shown in Table 2, a significant portion (43.2%) of the query graphs generated by DAC consists of more than one triple (i.e., option). According to the statistics, among those comprised of more than two triples, 39.4% contain functional operations and logical disjunction. We notice that the performance on WebQ does not drop a lot as questions become more complex. The stability demonstrates the effectiveness and robustness of DAC at capturing the compositional semantics of complex questions. We also notice that DAC mainly generates query graphs with only one triple for unanswerable questions, e.g., the question "What happened to Jill Valentine?". This may explain the reason for low performance on "simple" questions.

4.4 Ablation study

The improvements on the benchmark datasets demonstrate that DAC is adept at complex question answering. To have a deep insight into the design of DAC, we perform model ablation studies on curriculum learning and updating the Critic.

Table 3: Training performance (%) on WebQ and CompQ-1 with and without curriculum learning.

Setting	WebQ			CompQ-1		
	Prec.	Rec.	F1	Prec.	Rec.	F1
w/	53.7	64.5	54.9	43.8	55.0	45.2
w/o	49.2	59.0	50.1	38.7	49.5	40.6

Curriculum learning. As is shown in Table 3, the proposed model fits the training data better when applying curriculum learning, in which the agent first starts out with only easy questions and then gradually increase the question complexity. Such a setting essentially relies on a notion of "easy" and "hard" examples, and in our formulation, questions can be divided based on the good query graphs generated by hand-crafted rules. We utilize this distinction to teach the agent how to generalize easier query graphs before harder ones. With curriculum learning, the agent can gradually handle those questions, for which the hand-crafted rules cannot generate high-reward query graphs.

Updating the Critic. As is shown in Algorithm 1, every several epochs, query graphs generated by the model that achieve higher F1 scores are added to train a better Critic. It is of great importance for the Critic to provide the Actor with proper intrinsic motivations. To evaluate the primitive actions correctly, we pre-train the Critic with query graphs generated by hand-crafted rules. However, fixing the parameters in the Critic seems to get the model stuck in a local optimum and produce worse results. As is illustrated in Figure 4, with respect to iterations, the changes in test sets avg F_1 of DAC (red) and the variation (blue) generally show that the strategy can help the model converge to a better performance level.

4.5 Error Analysis

We provide the error analysis of our approach on the WebQ dataset. Specifically, we randomly select 100 questions where no correct answers are returned. There are mainly five reasons for the failure of these questions.

The first reason is the failure of constant node linking, especially entity linking, and this error occurs due to the high ambiguity of entity mentions. For example, the question "What channel is the USA Pageant on?" expects the broadcast information of "Miss USA Pageant", but S-MART selects the nation "USA" as the topic entity, making the gold answer impossible to be returned.

The second and the most common reason is the reasoning error. This type of error occurs when the model failed to understand the main semantics due to the variety of natural language. For example, the question "Who is Jamie Little engaged to" expects the spouse of Jamie Little as the answer, while our model returns her profession since that the phrase "engaged to" has both the two meanings.

Missing information in the KG is the third reason for the failure. This error means that the KG, i.e., Freebase, cannot support answering some questions. For example, the question "What country did Germany invade first in WW1?" requires sorting over the time when each country was invaded. However, our dump of Freebase stores nothing about the information.

The fourth reason is incorrect "gold" answers to some questions in the dataset. This error means that WebQ provides incorrect answers to some questions. For example, the question "When did the San Francisco earthquake occur?" expects the exact date of the 1906 San Francisco earthquake, and the answer should be "1906-04-18". However, the gold answer provided is "1906 San Francisco earthquake".

The last reason is that WebQ contains unanswerable questions. This type of error perhaps results from the semantic ambiguity. For example, we cannot figure out what the question "What happened to Jill Valentine?" means.

5 CONCLUSION

In this paper, we proposed a hierarchical query graph generation framework, namely Director-Actor-Critic (DAC), for complex question answering over knowledge graphs. The Director selects an option to determine which type of triple is needed. Then the Actor selects primitive actions to add the chosen type of triple into the query graph, and the Critic calculates a semantic score between the newly generated triple and the given question. Different from the previous methods, our framework does not limit the answer nodes to the topic-entity-centric subgraphs. Triples, functional operations, and logical connection types are generated by a trainable model instead of heuristic rules.

To train from weak supervision, we based DAC on hierarchical Reinforcement Learning (RL) with intrinsic motivation. Meanwhile, DAC employs curriculum learning, in which the agent first starts out with only easy questions and then gradually increases the question complexity. Moreover, every several epochs, better query graphs generated by the model are added to train a better Critic, avoiding the model stuck in a local optimum. We evaluated the effectiveness of the proposed DAC on widely-used benchmark datasets, and our approach outperforms all the baselines.

In future work, we plan to study the follow-up problems: DAC and the previous methods both rely on the result of constant node linking, which is generated by an external linking tool. How to fix the mistakes made by the linking tool when training our model? Meanwhile, mapping question utterances to KG relations is still challenging due to the variety of natural languages.

ACKNOWLEDGMENTS

This work is supported by the National Key Research and Development Program of China under grants 2018YFB1402600 and 2017YFC0820404, the National Natural Science Foundation of China under grants U1836206, U1911401, 61772501, the GF Innovative Research Program, and Central China Talents Program - Leading Talents Project of Science and Technology Innovation.

REFERENCES

[1] Abdalghani Abujabal, Mohamed Yahya, Mirek Riedewald, and Gerhard Weikum. 2017. Automated Template Generation for Question Answering over Knowledge Graphs. In *Proceedings of the 26th International Conference on World Wide Web*. 1191–1200.

[2] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*. Springer, 722–735.

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

[4] Junwei Bao, Nan Duan, Zhao Yan, Ming Zhou, and Tiejun Zhao. 2016. Constraint-based question answering with knowledge graph. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics*. 2503–2514.

[5] Hannah Bast and Elmar Haussmann. 2015. More Accurate Question Answering on Freebase. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15)*. 1431–1440.

[6] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*. 41–48.

[7] Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. 1533–1544.

[8] Jonathan Berant and Percy Liang. 2014. Semantic Parsing via Paraphrasing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland, 1415–1425.

[9] Nikita Bhutani, Xinyi Zheng, and H V Jagadish. 2019. Learning to Answer Complex Questions over Knowledge Bases with Query Composition. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*. 739–748.

[10] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 1247–1250.

[11] Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question Answering with Subgraph Embeddings. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 615–620.

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

[13] 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*. Springer, 165–180.

[14] Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1724–1734.

[15] Li Dong, Furu Wei, Ming Zhou, and Ke Xu. 2015. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Vol. 1. 260–269.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

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

[18] 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 (2018), 824–837.

[19] Sen Hu, Lei Zou, and Xinbo Zhang. 2018. A State-transition Framework to Answer Complex Questions over Knowledge Base. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2098–2108.

[20] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

[21] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. 2016. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*. 3675–3683.

[22] Chen Liang, Jonathan Berant, Quoc Le, Kenneth D Forbus, and Ni Lao. 2017. Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics*. 23–33.

[23] Kangqi Luo, Fengli Lin, Xusheng Luo, and Kenny Zhu. 2018. Knowledge Base Question Answering via Encoding of Complex Query Graphs. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2185–2194.

[24] A.L. Maas, A.Y. Hannun, and A.Y. Ng. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *Proceedings of the International Conference on Machine Learning*. Atlanta, Georgia.

[25] Gaurav Maheshwari, Priyansh Trivedi, Denis Lukovnikov, Nilesh Chakraborty, Asja Fischer, and Jens Lehmann. 2019. Learning to rank query graphs for complex question answering over knowledge graphs. In *International Semantic Web Conference*. Springer, 487–504.

[26] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*. 55–60. http://www.aclweb.org/anthology/P/P14/P14-5010

[27] Alexander Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. 2016. Key-Value Memory Networks for Directly Reading Documents. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 1400–1409.

[28] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[29] Baolin Peng, Xiujun Li, Lihong Li, Jianfeng Gao, Asli Celikyilmaz, Sungjin Lee, and Kam-Fai Wong. 2017. Composite Task-Completion Dialogue Policy Learning via Hierarchical Deep Reinforcement Learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2231–2240.

[30] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 1532–1543.

[31] Eric Prudhommeaux. 2008. SPARQL query language for RDF. *http://www. w3. org/TR/rdf-sparql-query/* (2008).

[32] Yunqi Qiu, Yuanzhuo Wang, Xiaolong Jin, and Kun Zhang. 2020. Stepwise Reasoning for Multi-Relation Question Answering over Knowledge Graph with Weak Supervision. In *Proceedings of the 13th International Conference on Web Search and Data Mining (WSDM '20)*. 474–482.

[33] Siva Reddy, Mirella Lapata, and Mark Steedman. 2014. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics* 2 (2014), 377–392.

[34] Amrita Saha, Ghulam Ahmed Ansari, Abhishek Laddha, Karthik Sankaranarayanan, and Soumen Chakrabarti. 2019. Complex Program Induction for Querying Knowledge Bases in the Absence of Gold Programs. *Transactions of the Association for Computational Linguistics* 7 (2019), 185–200.

[35] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *The Semantic Web*. 593–607.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.

[37] Satinder P Singh, Andrew G Barto, and Nuttapong Chentanez. 2005. Intrinsically motivated reinforcement learning. In *Advances in neural information processing systems*. 1281–1288.

[38] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 697–706.

[39] Haitian Sun, Bhuwan Dhingra, Manzil Zaheer, Kathryn Mazaitis, Ruslan Salakhutdinov, and William Cohen. 2018. 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*. 4231–4242.

[40] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*.

[41] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[43] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3 (1992), 229–256.

[44] Kun Xu, Siva Reddy, Yansong Feng, Songfang Huang, and Dongyan Zhao. 2016. Question Answering on Freebase via Relation Extraction and Textual Evidence. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. 2326–2336.

[45] Yi Yang and Ming-Wei Chang. 2015. 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*. 504–513.

[46] Rui Ye, Xin Li, Yujie Fang, Hongyu Zang, and Mingzhong Wang. 2019. A Vectorized Relational Graph Convolutional Network for Multi-Relational Network Alignment. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. 4135–4141.

[47] 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 *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, Vol. 1. 1321–1331.

[48] Pengcheng Yin, Nan Duan, Ben Kao, Junwei Bao, and Ming Zhou. 2015. Answering Questions with Complex Semantic Constraints on Open Knowledge Bases. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 1301–1310.

[49] Shuguang Zhu, Xiang Cheng, and Sen Su. 2020. Knowledge-based question answering by tree-to-sequence learning. *Neurocomputing* 372 (2020), 64–72.

[50] Lei Zou, Ruizhe Huang, Haixun Wang, Jeffrey Xu Yu, Wenqiang He, and Dongyan Zhao. 2014. Natural Language Question Answering over RDF: A Graph Data Driven Approach. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*. 313–324.