3-2020

Using knowledge bases for question answering

Yunshi LAN
Singapore Management University, yslan.2015@phdis.smu.edu.sg

USING KNOWLEDGE BASES FOR QUESTION ANSWERING

YUNSHI LAN

SINGAPORE MANAGEMENT UNIVERSITY

2020

Using Knowledge Bases for Question Answering

by

Yunshi Lan

Submitted to School of Information Systems in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in Computer Science

Dissertation Committee:

Jing JIANG (Supervisor / Chair)
Associate Professor of Information Systems
Singapore Management University

Feida ZHU (Co-supervisor)
Associate Professor of Information Systems
Singapore Management University

David LO
Associate Professor of Information Systems
Singapore Management University

Lun-Wei KU
Associate Research Professor
Institute of Information Science, Academia Sinica

Singapore Management University
2020

I hereby declare that this PhD dissertation is my original work
and it has been written by me in its entirety.
I have duly acknowledged all the sources of information
which have been used in this dissertation.

This PhD dissertation has also not been submitted for any degree
in any university previously.

Yunshi lan

Yunshi Lan
27 Feb 2020

Using Knowledge Bases for Question Answering

Yunshi Lan

Abstract

A knowledge base (KB) is a well-structured database, which contains many of entities and their relations. With the fast development of large-scale knowledge bases such as Freebase [9], DBpedia [1] and YAGO [67], knowledge bases have become an important resource, which can serve many applications, such as dialogue system, textual entailment, question answering and so on. These applications play significant roles in real-world industry.

In this dissertation, we try to explore the entailment information and more general entity-relation information from the KBs. Recognizing textual entailment (RTE) is a task to infer the entailment relations between sentences. We need to decide whether a hypothesis can be inferred from a premise based on the text of two sentences. Such entailment relations could be potentially useful in applications like information retrieval and commonsense reasoning. It's necessary to develop automatic techniques to solve this problem. Another task is knowledge base question answering (KBQA). This task aims to automatically find answers to factoid questions from a knowledge base, where answers are usually entities in the KB. KBQA task has gained much attention in recent years and shown promising contribution to real-world problems. In this dissertation, we try to study the applications of knowledge bases in textual entailment and question answering:

- We propose a general neural network based framework which can inject lexical entailment relations to RTE, and a novel model is developed to embed lexical entailment relations. The experiment results show that our method can benefit general textual entailment model.

- We design a KBQA method based on an existing reading comprehension

model. This model achieves competitive results on several popular KBQA datasets. In addition, we make full use of contextual relations of entities in the KB. Such enriched information helps our model to attain state-of-art.

- We propose to perform topic unit linking where topic units cover a wider range of units of a KB. We use a generation-and-scoring approach to gradually refine the set of topic units. Furthermore, we use reinforcement learning to jointly learn the parameters for topic unit linking and answer candidate ranking in an end-to-end manner. Experiments on three commonly used benchmark datasets show that our method consistently works well and outperforms the previous state of the art on two datasets.

- We further investigate multi-hop KBQA task, i.e., question answering from KB where questions involve multiple hops of relations, and develop a novel model to solve such questions in an iterative and efficient way. The results demonstrate that our method consistently outperforms several multi-hop KBQA baselines.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Knowledge bases (KB) are structured databases. They encode various kinds of knowledge inside. Knowledge bases are also known as knowledge graphs, where each node represents an entity and edges represent relations between entities. Since usually relations have directions, we consider knowledge graphs to be directed graphs.

There are diverse knowledge bases. Freebase [9], DBpedia [1] and YAGO [67] are knowledge bases with entities and their relations in general domains. Figure 1.1 is a sub-graph of Freebase. We can see that the entity “Mike Kelly (American baseball player)” is connected with another entity “USA” via the relation “nationality”. This relation is represented by a triplet *(Mike Kelly (American baseball player), nationality, USA)*, where we say “Mike Kelly (American baseball player)” is the subject or head entity, “nationality” is the relation and “USA” is the object or tail entity. This triplet states the fact that “the nationality of Mike Kelly (American baseball player) is USA”. In addition to these knowledge bases that store general knowledge about real-world entities and their relations, there are also some other knowledge bases that contain basic concepts such as words. Examples of such knowledge bases include WordNet [51], ConceptNet [43], BabelNet [54], SenticNet [16] and HowNet [23]. Take WordNet for example. It is a large hierarchical knowledge base encoding various kinds of relations among words. Figure 1.2 shows a sub-graph

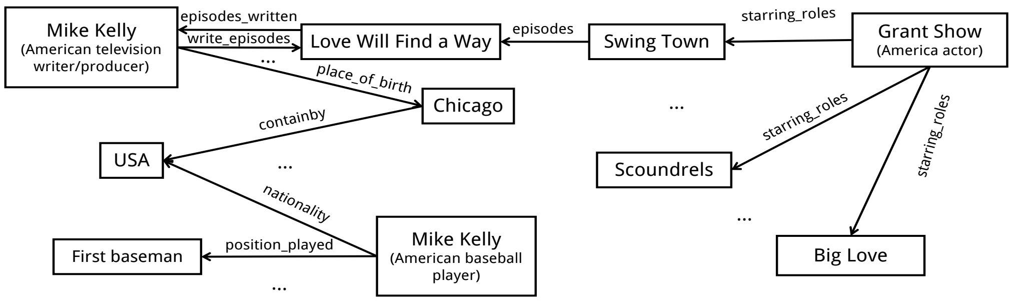


Figure 1.1: A sub-graph of Freebase

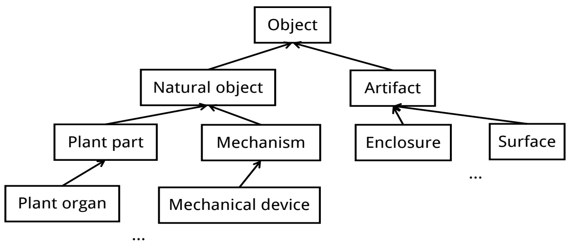


Figure 1.2: A sub-graph of WordNet

of WordNet. The entities “Object”, “Natural object”, “Artifact”, etc. are semantic concepts and they are interconnected via the entailment relation “hypernym”. The triplet *(Artifact, hypernym, Object)* indicates that “Object” is the hypernym of the word “Artifact”. We can see knowledge stored in such a structured form is relatively easy to maintain and to query.

There are many applications that can benefit from knowledge bases. For example, when we create a dialogue system for flight reservations, the flight data is stored as a knowledge base. After analyzing users’ requests and generating queries, we can retrieve corresponding flight information from the knowledge base that stores all the data related to flight reservations. Such information should be incorporated into a dialogue generator and the natural language responses can be sent back to the users. Similarly, other applications like recommendation and question answering can also take advantage of knowledge bases.

In this dissertation, we focus on two problems which can benefit from using knowledge stored in knowledge bases: textual entailment, and knowledge base question answering. For Recognizing textual entailment (RTE), we managed to

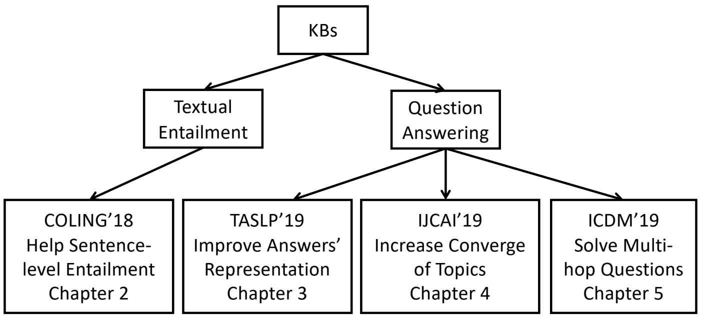

KBs
Textual Entailment
Question Answering
COLING'18 Help Sentence-level Entailment Chapter 2
TASLP'19 Improve Answers' Representation Chapter 3
IJCAI'19 Increase Converge of Topics Chapter 4
ICDM'19 Solve Multi-hop Questions Chapter 5

Figure 1.3: The organization of the topics in this dissertation.

Premise	Hypothesis	Ground Truth
Multiple males are playing a soccer game.	Some men are playing a sport.	Entailment
A man inspects the uniform of a figure.	The man is sleeping.	Contradiction
An older and younger man smiling.	Two men are and laughing and the cats playing.	Neutral

Table 1.1: Three examples from SNLI dataset.

inject entailment information from the KB to sentence-level entailment. For knowledge base question answering (KBQA), we solved three independent but related sub-problems. In Figure 1.3, we summarize what we have done with the KBs.

1.1 Using WordNet for Textual Entailment

RTE aims to identify the entailment relation between two sentences. We call these two sentences premise and hypothesis, respectively. We give three examples from the SNLI dataset [13] in Table 1.1. In the first example, we know if the premise is true, the hypothesis must be true, so their entailment label is "Entailment". In the second example, we know if the premise is true, then the hypothesis cannot be true. Thus their entailment label is "Contradiction". In the last example, since we can't find an obvious relation between these two sentences, we annotate their entailment relation as "Neutral".

General methods try to learn the entailment patterns from the training datasets via advanced neural networks. Such methods are powerful to capture general sentence-level as well as word-level entailment patterns and have shown promising results on multiple RTE datasets. However, some lexical entailment knowledge is missing from the training data. In particular, if the test data contains a great number of unseen lexical entailment patterns, it would be difficult to make the right predictions. For the first example in Table 1.1, if “soccer game” and “sport” never occur in training set, we cannot infer the relation of this premise and hypothesis. On the other hand, if we have seen a triplet *(Soccer game, hypernym, Sport)* in a knowledge base, we can easily infer the entailment relation for the above example. Therefore, we attempt to inject lexical knowledge into textual entailment under the architecture of neural networks, which rarely been explored by current works [37]. We think this could help to improve the efficiency and interpretability of the neural network model.

1.2 Question Answering from Knowledge Bases

KBQA aims to return entities that could serve as correct answers for given natural language questions. Given the sub-graph in Freebase of Figure 1.1, we ask one question “What episode was Mike Kelly the writer of”. Then we need to find the correct triplet from the whole knowledge graph, which expresses the same relation of the question. In this example, the triplet *(Mike Kelly (American television writer/producer), write_episodes, Love Will Find a Way)* represents exactly what the question asks, so the object of the triplet “Love Will Find a Way” should be retrieved as the predicted answer.

The main challenge of KBQA task is that from a huge knowledge base with millions of triplets, finding the matched triplet is not trivial, especially when the question is ambiguous. A popular method to solve KBQA task is that we can match all possible relation paths with questions and the object of the relation path with the highest matching score will be deemed as the predicted answer. In above example, the triplets *(Mike Kelly (American television writer/producer), write_episodes, Love Will Find a Way)*, *(Mike Kelly (American television writer/producer), place_of_birth, Chicago)* in Table 1.1 are possible relation paths to answer the question. The correct answer should be selected among “Love Will Find a Way” and “Chicago”. While these techniques have shown good results on KBQA datasets, there are still some flaws in them. When neural networks are applied to the matching part, the state-of-art model has not been used. So we plan to adopt a more finely-designed model to the KBQA system and this model could improve the performance of matching [38]. In addition, to form the features of candidate answers, besides the relation paths, the contextual relations of answers could also be included. These additional features will enrich the representations of candidate answers and bring benefit to more accurate predictions.

Predicting the relation path is a crucial step in KBQA. Besides, the entity linking is the first and important step of the overall pipeline. In the above example, we need to link the mentioned span “Mike Kelly” in the question to the entity “(Mike Kelly (American television writer/producer))”, so that we could retrieve the possible relation paths from the KBs. However, linking the question to the correct entities is not easy. Existing work usually leverages existing entity linking tools to achieve the linking step. However, such entity linking tools are not supervised via the KBQA task, we may encounter the cases where the correct entities are ranked low, or the correct entities are missed out entirely. To solve this problem, we propose a model to involve in more topic units, including entities, relations or types from the KBs, which could increase the recall of the answers. We first select those topic units, then retrieve and rank the relation paths from the KBs. We further connect these two steps via the Reinforcement Learning algorithm, which could improve the overall results [39].

The above method demonstrates effectiveness in answering simple questions. However, for more complicated questions, it may be not good enough. Therefore, we turn our attention to multi-hop KBQA, where some inferences are necessary to be included to answer the questions. Still, for Figure 1.1, a possible multi-hop question could be “Which country is the birthplace of the writer of episode Love Will Find a Way located in”. To answer such a question, we can start from the entity

"Love Will Find a Way", but only looking for relations one or two hops away from "Love Will Find a Way" might be limited. More triplets should be involved as the relation paths. However, the relation paths would be extended a lot, which leads to the explosive number of candidate answers and features. This gives new challenges to KBQA task. Therefore, we propose a novel model to retrieve the correct entities iteratively [40]. After each iteration, we will prune unrelated entities. This will not only save us from overwhelming computation and memory demands but also make it easier to differentiate the correct relation path from the candidate relation paths on a small scale. Furthermore, the model performs well in checking whether a question has been answered completely and deciding the moment to stop iterations.

1.3 Dissertation Structure

As shown in Figure 1.3, for the remaining parts of the dissertation, we summarise them into two parts. For Part I, which only contains a single Chapter 2, we first review the related work and present our work on embedding WordNet knowledge for textual entailment. Specifically, we propose a novel model to encode the lexical entailment relations and try to inject the lexical knowledge from a knowledge base to the textual entailment task. For Part II, we first review the historical work on knowledge base question answering. Next, Chapter 3 covers our work on knowledge base question answering with a matching-aggregation model and question-specific additional relations. In this work, we develop a KBQA model and make full use of connected relations of candidate answers. Then we introduce our work on KBQA with topic units in Chapter 4. In this work, besides topic entities, we involve more units in the KBs to increase the recall of the answers. We further apply the Reinforcement Learning to select the correct units at the first stage, retrieve and rank the candidate paths at the second stage. In Chapter 5, we focus on multi-hop KBQA task and propose an iterative sequence matching model for multi-hop knowledge base question answering. This model will try to solve this problem by matching entities iteratively. Finally, we talk about future directions in Chapter 6.

Part I

Chapter 2

Embedding WordNet Knowledge for Textual Entailment

2.1 Introduction

Recognizing textual entailment (RTE) is the task of determining whether a hypothesis sentence can be inferred from a given premise sentence. The task has been well studied since it was first introduced by Dagan et al. [20]. Recently, there has been much interest in applying deep learning models to RTE [13, 62, 73, 56, 65]. These models usually do not perform any linguistic analysis or require any feature engineering but have been shown to perform very well on this task.

Intuitively, lexical-level entailment relations should help sentence-level RTE. For example, Table 2.1 shows a premise and a hypothesis taken from the test data of the SICK dataset [47]. We can see that in this example the premise entails the hypothesis, and in order to correctly identify this relation, one has to know that the word *kettle* entails the word *pot*. However, if we train a neural network model on a set of labeled sentence pairs, and if the training dataset does not contain the word pair *kettle* and *pot* anywhere, it would be hard for the learned model to know that *kettle* entails *pot* and subsequently predict the relation between the premise and the hypothesis to be *entailment*. On the other hand, from WordNet we can easily find out that *pot* is a direct hypernym of *kettle* and therefore *kettle* should entail *pot*. If this kind of prior knowledge can be injected into a trained RTE model, then for sentence pairs with words that do not occur in the training data but can be found in WordNet, the RTE predictions could potentially be made easier.

Premise: *Someone is stirring chili in a* ***kettle****.*
Hypothesis: *Someone is stirring chili in a* ***pot****.*

Table 2.1: An example of a pair of premise and hypothesis from SICK.

Indeed, WordNet [51] knowledge has been used to help RTE in a number of previous studies [19, 5, 48]. However, most of these previous studies were not based on neural network models, and thus their base models without using WordNet may not represent the state of the art or may require much human effort. For example, our baseline model based on a neural network model without using any WordNet knowledge can achieve an accuracy of 84.1% on the SICK test data, but the baseline model by Martinez-Gomez et al. [48], which uses logical semantic representations and a theorem prover, can only achieve an accuracy of 76.7%. Given the advantages of deep learning approaches to RTE, it is therefore desirable to incorporate WordNet knowledge into deep learning based solutions to RTE. In this chapter of the dissertation, we are going to investigate how WordNet knowledge could be easily brought into neural network models. This could help to improve the accuracy of the textual entailment and increase interpretation of the neural network models.

We propose to first embed lexical entailment knowledge contained in WordNet into word vectors. We call these special word vectors “entailment vectors.” We then incorporate these entailment vectors into a recently proposed decomposable attention model for RTE. To learn these entailment vectors that encode lexical entailment relations, we can use a standard neural network. We also propose a set-theoretic model that has better theoretical justification. Our experiments show that using these entailment vectors learned from WordNet can indeed significantly improve the performance of RTE on the SICK dataset and the SNLI dataset. The performance of our method is also better compared with the state of the art on SICK.

2.2 Literature Review

Recently, neural-network-based approaches have been developed to learn vector representations of words [49, 57]. These word embeddings are trained from an unlabeled large corpus and for general usage. In contrast, our entailment word vectors are not meant to be used as general-purpose word embeddings. They are designed specifically for incorporating external knowledge from WordNet into neural network models for RTE. Although there has been some work attempting to inject external lexical knowledge into word embeddings [46, 17], these learned word embeddings don't contain lexical entailment information specifically for RTE. To encode the lexical entailment relations, some existing methods for lexical entailment are based on the distributional inclusion hypothesis and leverage co-occurrence information of words from a large corpus [33, 26, 34, 8]. In contrast, our first work uses labeled word pairs derived from WordNet to learn entailment vectors to encode the lexical entailment relations. The semantic relations we consider are also designed specifically for natural language inference.

Similar to our motivation, some studies have also attempted to inject lexical knowledge into the RTE task. [2] proposed compositional distributional models to extend representations from words to sentences. By mapping from a premise to a hypothesis by dependency trees or proposition extraction and then comparing lexical entailment between nodes or words, [30] and [55] tried to solve RTE via WordNet. [4], [5] and [48] first changed sentences to logical forms and then leveraged logic inference engines combined with lexical entailment axioms to make entailment judgments. However, to our best knowledge, we are rarely aware of the work on how to apply lexical knowledge to textual entailment based on neural networks.

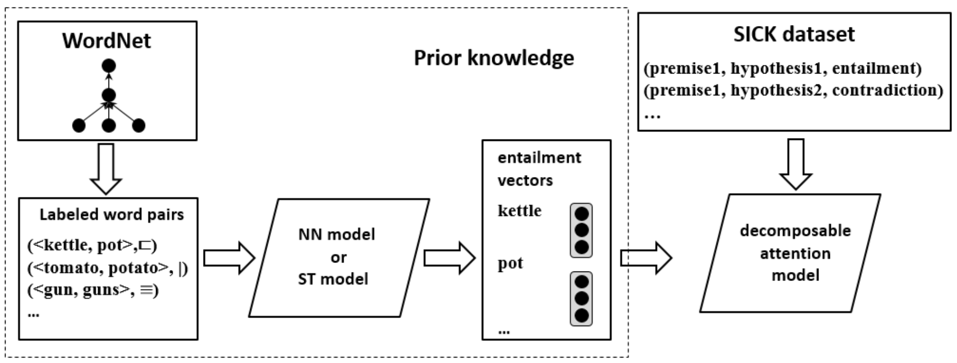


Figure 2.1: An overview of our approach.

2.3 Method

In this section, we present our method that learns the entailment vectors that encode prior knowledge of lexical entailment relations. We also present how we incorporate these entailment vectors into a neural network model for RTE. The overall framework of our method is illustrated in Figure 2.1.

2.3.1 Learning Entailment Vectors

We assume that our prior knowledge of lexical entailment relations is contained in word pairs and their entailment relations. Specifically, let $\mathcal{R}$ denote a set of lexical entailment relations. For example, $\mathcal{R}$ may contain *entailment* and *reverse-entailment*. Let $\mathcal{L} = \{(w_{i,1}, w_{i,2}, r_i)\}_{i=1}^{N}$ denote a set of N word pairs together with their relation labels, where $w_{i,1}$ and $w_{i,2}$ are two words and $r_i \in \mathcal{R}$. For example, $\mathcal{L}$ may contain the triplet (<*pot*, *kettle*>, *reverse-entailment*). Let $\mathcal{V}$ denote the set of all unique words found in $\mathcal{L}$.

In order to encode the lexical entailment knowledge contained in $\mathcal{L}$, we propose to learn a dense vector for each word $w \in \mathcal{V}$ such that the entailment relation between two words in $\mathcal{V}$ can be easily detected from their word vectors. We refer to these dense word vectors as "entailment vectors." Note that these entailment vectors are different from commonly-used word embeddings such as GloVe and word2vec, because the entailment vectors are not learned from a large corpus and thus do not

Name	Symbol	Set-theoretic definition	Example
(strict) entailment	$x_1 \sqsubset x_2$	$x_1 \subset x_2$	woman, person
(strict) reverse entailment	$x_1 \sqsupset x_2$	$x_1 \supset x_2$	person, woman
equivalence	$x_1 \equiv x_2$	$x_1 = x_2$	couch, sofa
alternation	$x_1 \mid x_2$	$x_1 \cap x_2 = \emptyset \wedge x_1 \cup x_2 \neq \mathcal{D}$	woman, man
negation	$x_1 \wedge x_2$	$x_1 \cap x_2 = \emptyset \wedge x_1 \cup x_2 = \mathcal{D}$	able, unable
cover	$x_1 \smile x_2$	$x_1 \cap x_2 \neq \emptyset \wedge x_1 \cup x_2 = \mathcal{D}$	person, non-woman
independence	$x_1 \# x_2$	(else)	woman, doctor

(a)

	$c_{0,0}$	$c_{0,1}$	$c_{1,0}$	$c_{1,1}$
$x_1 \sqsubset x_2$	$-$	>0	$=0$	$-$
$x_1 \sqsupset x_2$	$-$	$=0$	>0	$-$
$x_1 \equiv x_2$	$-$	$=0$	$=0$	$-$
$x_1 \mid x_2$	>0	$-$	$-$	$=0$
$x_1 \wedge x_2$	$=0$	$-$	$-$	$=0$
$x_1 \smile x_2$	$=0$	$-$	$-$	>0
$x_1 \# x_2$	>0	$-$	$-$	>0

(b)

Table 2.2: (a) Seven basic semantic relations defined by Bill et al. [44]. Here $\mathcal{D}$ is the universe that contains all entities. Each x_1 (or x_2) is a language unit that represents a subset of $\mathcal{D}$. (b) Criteria for different basic semantic relations based on the counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$.

encode the distributional properties of words. They are learned from labeled word pairs in $\mathcal{L}$. We will show later that they can be used in combination with common word embeddings such as word2vec to help RTE.

In what follows, we first describe what lexical entailment relations we include in $\mathcal{R}$. We then present two different neural network models used to learn the entailment vectors from $\mathcal{L}$. We defer the description of how we derive labeled word pairs $\mathcal{L}$ from WordNet until Section 2.3.3.

Lexical Entailment Relations

Recall that eventually the entailment vectors will be used for textual entailment. In standard textual entailment datasets such as SICK and SNLI, there are three sentence-level entailment relations: *entailment*, *contradiction* and *neutral*. However, at word-level the entailment relations are different.

In a previous study, MacCartney et al. [44] developed a framework for natural language inference. As the basis of their framework, they defined seven basic semantic relations between language units, which we show in Table 2.2a. These relations cover all the possible relations between two language units x_1 and x_2. We can see that each relation has an equivalent set-theoretic definition, which dates back to Montague Semantics [32]. Each language unit x_1 or x_2 is modeled as a set, and is supposedly a subset of the universe $\mathcal{D}$. Therefore their relations can be determined by the relation between the two sets. For example, if the language units we consider are nouns, we can regard $\mathcal{D}$, the universe, as the set of all entities in the world. If x_1 is the word "person," we can regard x_1 as a set that contains all people. If x_2 is the word "woman," we can regard x_2 as a set that contains all people who are female. Since x_1 is a superset of x_2, based on the definition, x_1 reversely entails x_2, or $x_1 \sqsupset x_2$. This makes sense because "person" reversely entails "woman," or in other words, "woman" entails "person."

In this work, we only use a subset of these relations, namely, *entailment* ($\sqsubset$), *reverse-entailment* ($\sqsupset$), *alternation* ($|$) and *equivalence* ($\equiv$). In other words, $\mathcal{R} = \{\sqsubset, \sqsupset, |, \equiv\}$. We do not include *negation*, *cover* or *independence* because we find it hard to automatically derive word pairs with these relations from WordNet.

Standard Neural Network Model

To learn entailment vectors from $\mathcal{L}$, we first consider a standard neural network model. Let $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ denote the entailment vectors of two words, which we are trying to learn, where d is the number of dimensions of the vectors. We can combine the two vectors into a single vector as follows:

$$\mathbf{h} = \tanh(\mathbf{M} \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} + \mathbf{b}),$$

where $\mathbf{M} \in \mathbb{R}^{l \times 2d}$ is a weight matrix, $\mathbf{b} \in \mathbb{R}^l$ is a weight vector, $\tanh(\cdot)$ is applied element-wise to the vector, $\mathbf{h} \in \mathbb{R}^l$ is a hidden vector and l is the number of dimensions of the hidden vector.

The hidden vector $\mathbf{h}$ can then go through a linear transform followed by a softmax layer to be used to predict the relation label r between the two words:

$$p(r \mid \mathbf{h}) = \text{softmax}(\mathbf{M}'\mathbf{h} + \mathbf{b}'), \tag{2.1}$$

where $\mathbf{M}' \in \mathbb{R}^{k \times l}$ and $\mathbf{b}' \in \mathbb{R}^{k}$ denote a weight matrix and a bias vector, and $k = |\mathcal{R}|$.

Given all the labeled word pairs in $\mathcal{L}$, we can use the cross entropy loss as the objective function to learn the entailment vector for each $w \in \mathcal{V}$ as well as the various parameters above.

Set-Theoretic Model

Although the standard neural network model above is straightforward, it is hard to explain how the learned word vectors can encode the lexical entailment relations. Inspired both by the set-theoretic definitions of the basic semantic relations shown in Table 2.2a and by some recent work on formal distributional semantics [28, 61], we hypothesize that a good entailment vector for a word essentially encodes which elements in $\mathcal{D}$ are members of the set representing this word. We now present a novel set-theoretic model to learn the entailment vectors.

To illustrate our idea, we first show that in the extreme case when word vectors we try to learn are binary vectors precisely representing set memberships, the semantic relation between two words can be determined by comparing the two vectors element-wise. Specifically, let D denote the size of the universe $\mathcal{D}$. Since a word x can be regarded as a subset of $\mathcal{D}$, we assume that the entailment vector of x is a D-dimensional binary vector $\mathbf{x}$, where $\mathbf{x}_i$ is 1 if the i^{th} element in $\mathcal{D}$ is inside the set x and 0 otherwise. Given two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ representing two words, in order to determine the relationship between them, we need to check the relationship between the two sets. To do so, we define the following counters for $p, q \in \{0, 1\}$:

$$c_{p,q} = \sum_{i=1}^{D} \delta(x_{1,i}, p)\delta(x_{2,i}, q),$$

where $\delta(s, t)$ is 1 if s is equal to t and 0 otherwise. Essentially if we compare $\mathbf{x}_1$ and $\mathbf{x}_2$ element-wise, then $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$ count the number of times we see (0, 0), (0, 1), (1, 0) and (1, 1), respectively. It is not hard to show that the seven basic semantic relations in Table 2.2a correspond to different values of these $c_{p,q}$. For example, if $c_{0,1}$ is 0 and $c_{1,0}$ is 0, then the two sets (words) are equivalent. If $c_{0,1}$ is positive and $c_{1,0}$ is 0, then $\mathbf{x}_1$ is a subset of $\mathbf{x}_2$, and therefore the first word entails the second word. Table 2.2b shows the criteria for each basic semantic relation based on the values of these counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$.

We can use a simple example to illustrate the idea above. Suppose the universe contains four elements: $\mathcal{D} = \{\textit{dog}, \textit{cat}, \textit{tiger}, \textit{computer}\}$. The word *animal* is a set that contains *dog*, *cat* and *tiger*, and thus can be represented by the vector $[1, 1, 1, 0]$. The word *pet* should contain only *dog* and *cat*, so it can be represented by the vector $[1, 1, 0, 0]$. In this case, the values of the four counters are the following: $c_{0,0} = 1$, $c_{0,1} = 0$, $c_{1,0} = 1$, and $c_{1,1} = 2$. According to Table 2.2b, we can determine that *animal* reversely entails *pet* based on the values of these counters.

Generally speaking, however, the number of elements in the universe is huge and therefore it is not feasible to learn word vectors that precisely represent the memberships of all these elements. In the following model we propose, we do not strictly force the entailment vectors to be binary. We also introduce a hidden layer that is deterministically derived from the entailment vectors. This hidden layer essentially represents $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$, and it is used for the prediction of the relation between two words.

Specifically, let $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ represent the entailment vectors corresponding to words w_1 and w_2.

Next, we introduce two complementary vectors:

$$\bar{\mathbf{w}}_1 = \mathbf{1} - \mathbf{w}_1, \qquad \bar{\mathbf{w}}_2 = \mathbf{1} - \mathbf{w}_2,$$

where $\mathbf{1}$ is a d-dimensional vector of 1s.

We then define a hidden vector $\mathbf{h}$ to be a 4-dimensional vector as follows:

$$\mathbf{h} = \frac{1}{d} \begin{bmatrix} \mathbf{w}_1^{\intercal}\mathbf{w}_2 & \mathbf{w}_1^{\intercal}\bar{\mathbf{w}}_2 & \bar{\mathbf{w}}_1^{\intercal}\mathbf{w}_2 & \bar{\mathbf{w}}_1^{\intercal}\bar{\mathbf{w}}_2 \end{bmatrix}^{\intercal}. \tag{2.2}$$

We can see that this hidden vector $\mathbf{h}$ roughly correspond to the four counters $c_{0,0}$, $c_{0,1}$, $c_{1,0}$ and $c_{1,1}$ defined above if the entailment vectors are binary vectors. But since we do not have the binary constraint, $\mathbf{h}$ actually contains real values rather than integer values. We also perform normalization by dividing the counts by d.

The same as shown in Eqn. (2.1), we can use $\mathbf{h}$ to predict the relation label between two words through a linear transform and a softmax layer. We can then again use the cross entropy loss as the objective function to learn the entailment vectors and the model parameters.

2.3.2 Using Entailment Vectors for RTE

Given the entailment vectors learned from $\mathcal{L}$ using either the standard neural network model or our proposed set-theoretic model, we can use them in a neural network model for textual entailment. In this dissertation we use a slightly modified version of the decomposable attention model proposed by Parikh et al. [56] because of its relative simplicity and good performance on the SNLI [13] dataset.

The setup of the textual entailment task is as follows. We are given two input sentences: a premise $X = ((\mathbf{x}_1, \mathbf{x}'_1), (\mathbf{x}_2, \mathbf{x}'_2), \ldots, (\mathbf{x}_m, \mathbf{x}'_m))$ and a hypothesis $Z = ((\mathbf{z}_1, \mathbf{z}'_1), (\mathbf{z}_2, \mathbf{z}'_2), \ldots, (\mathbf{z}_n, \mathbf{z}'_n))$, where each $\mathbf{x}_i$ (or $\mathbf{z}_j$) is a standard word embedding such as the word2vec embedding of the i^{th} (or j^{th}) word in the premise (or hypothesis), and $\mathbf{x}'_i$ (or $\mathbf{z}'_j$) is the newly-learned entailment vector of the i^{th} (or j^{th})

word in the premise (or hypothesis), as described in Section 2.3.1.[1] The goal is to predict a label $y \in \{\textit{entailment}, \textit{contradiction}, \textit{neutral}\}$ from X and Z.

The Modified Decomposable Attention Model

The decomposable attention model consists of the following three steps.

Attend: At this step, we derive attention weights from the word embeddings from X and Z. We first define

$$e_{ij} = F(\mathbf{x}_i)^\intercal F(\mathbf{z}_j),$$
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{j'=1}^{n} \exp(e_{ij'})}, \qquad \beta_{ij} = \frac{\exp(e_{ij})}{\sum_{i'=1}^{m} \exp(e_{i'j})},$$

where $F(\cdot)$ is a standard single-layer feed-forward neural network with ReLU activations [27]. We then define

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{n} \alpha_{ij} \cdot \mathbf{z}_j, \qquad \tilde{\mathbf{x}}'_i = \sum_{j=1}^{n} \alpha_{ij} \cdot \mathbf{z}'_j,$$
$$\tilde{\mathbf{z}}_j = \sum_{i=1}^{m} \beta_{ij} \cdot \mathbf{x}_i, \qquad \tilde{\mathbf{z}}'_j = \sum_{i=1}^{m} \beta_{ij} \cdot \mathbf{x}'_i.$$

$\tilde{\mathbf{x}}_i$ is a weighted version of the word embeddings from Z to match $\mathbf{x}_i$, and similarly $\tilde{\mathbf{x}}'_i$ is a weighted version of the entailment vectors from Z to match $\mathbf{x}'_i$. The same idea applies to $\tilde{\mathbf{z}}_j$ and $\tilde{\mathbf{z}}'_j$.

Compare: At this step, another single-layer feed-forward neural network G with ReLU activations is used to compare the aligned words. Normally, without considering the entailment vectors, the comparison is done as follows:

$$\mathbf{v}_{1,i} = G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i), \qquad \mathbf{v}_{2,j} = G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j),$$

where $\oplus$ is concatenation, and $\mathbf{v}_{1,i}$ and $\mathbf{v}_{2,j}$ represent the comparison results for the

[1]For those words which do not appear in $\mathcal{L}$ (the set of unique words in the training data), we randomly generate their entailment vectors in the range $(-0.1, 0.1)$.

i-th word in X and the j-th word in Z, respectively.

When we do consider the entailment vectors $\mathbf{x}'_i$ and $\mathbf{z}'_j$, as well as their counterparts $\tilde{\mathbf{x}}'_i$ and $\tilde{\mathbf{z}}'_j$ from the **Attend** step, we need to perform the comparison slightly differently. Here we use two different ways to perform the comparison, depending on whether the entailment vectors are learned using the standard neural network model or the set-theoretic model. If the standard neural network model is used to learn the entailment vectors, we simply concatenate all the vectors to perform comparison as follows:

$$\mathbf{v}_{1,i} = G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i \oplus \mathbf{x}'_i \oplus \tilde{\mathbf{x}}'_i), \qquad \mathbf{v}_{2,j} = G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j \oplus \mathbf{z}'_j \oplus \tilde{\mathbf{z}}'_j).$$

However, if the entailment vectors are learned using the set-theoretic model, because of the special properties of the entailment vectors as explained in Section 2.3.1, we perform comparison as follows:

$$\begin{aligned}
\mathbf{v}_{1,i} &= G(\mathbf{x}_i \oplus \tilde{\mathbf{x}}_i \oplus (\mathbf{x}'_i \odot \tilde{\mathbf{x}}'_i) \oplus (\mathbf{x}'_i \odot (1-\tilde{\mathbf{x}}'_i)) \oplus ((1-\mathbf{x}'_i) \odot \tilde{\mathbf{x}}'_i) \oplus ((1-\mathbf{x}'_i) \odot (1-\tilde{\mathbf{x}}'_i))),\\
\mathbf{v}_{2,j} &= G(\mathbf{z}_j \oplus \tilde{\mathbf{z}}_j \oplus (\mathbf{z}'_j \odot \tilde{\mathbf{z}}'_j) \oplus (\mathbf{z}'_j \odot (1-\tilde{\mathbf{z}}'_j)) \oplus ((1-\mathbf{z}'_j) \odot \tilde{\mathbf{z}}'_j) \oplus ((1-\mathbf{z}'_j) \odot (1-\tilde{\mathbf{z}}'_j))),
\end{aligned}$$

where $\odot$ is element-wise multiplication of two vectors.

Aggregate: Given $\mathbf{v}_{1,i}$ $(1 \le i \le m)$ and $\mathbf{v}_{2,j}$ $(1 \le j \le n)$ as defined above, we first use a standard LSTM model to process the two sequences separately, and then we use MaxPooling to aggregate the derived hidden vectors in order to obtain a single vector for each sequence:

$$\begin{aligned}
\mathbf{h}_{1,i} &= \text{LSTM}(\mathbf{v}_{1,i}, \mathbf{h}_{1,i-1}), & \mathbf{h}_{2,j} &= \text{LSTM}(\mathbf{v}_{2,j}, \mathbf{h}_{2,j-1}),\\
\mathbf{v}_1 &= \text{MaxPooling}(\mathbf{h}_{1,1}, \mathbf{h}_{1,2}, \ldots, \mathbf{h}_{1,m}), & \mathbf{v}_2 &= \text{MaxPooling}(\mathbf{h}_{2,1}, \mathbf{h}_{2,2}, \ldots, \mathbf{h}_{2,n}).
\end{aligned}$$

$$\mathbf{v}_{1,i} = \mathbf{v}_{1,i} \oplus \mathbf{v}'_{1,i}, \qquad \mathbf{v}_{2,j} = \mathbf{v}_{2,j} \oplus \mathbf{v}'_{2,j},$$

$$\mathbf{v}_1 = \text{Maxpooling}(\text{LSTM}(\mathbf{v}_{1,1},...,\mathbf{v}_{1,m})), \quad \mathbf{v}_2 = \text{Maxpooling}(\text{LSTM}(\mathbf{v}_{2,1},...,\mathbf{v}_{2,n})),$$

We then use $\mathbf{v}_1$ and $\mathbf{v}_2$ to make the final prediction:

$$p(y|\mathbf{v}_1,\mathbf{v}_2) = \text{softmax}(\mathbf{W}(\mathbf{v}_1 \oplus \mathbf{v}_2) + \mathbf{c}),$$

where $\mathbf{W}$ and $\mathbf{c}$ are parameters to be learned.

2.3.3 Implementation Details

In this section, we describe how we derive labeled word pairs from WordNet to construct $\mathcal{L}$. Because eventually we will test the learned entailment vectors on the SICK and SNLI textual entailment datasets, we focus on extracting word pairs with at least one word appearing in SICK or SNLI.

Set-theoretic model follows the semantic nature of the nouns. And involving different parts of speech could lead to multiple universe, which leads to difficulty of interpretation. Therefore, we only consider lexical relationship between nouns in our dissertation and filter other verbs, adjectives, adverbials etc. We follow the following steps to construct $\mathcal{L}$.

- If the synset of w_1 is a hypernym of the synset of w_2, we add $(w_1,w_2,\sqsupset)$ and $(w_2,w_1,\sqsubset)$ to $\mathcal{L}$.
- If w_1 and w_2 are in the same synset, or w_1 (w_2) is the plural form of w_2 (w_1), we add $(w_1,w_2,\equiv)$ and $(w_2,w_1,\equiv)$ to $\mathcal{L}$.
- If the synsets of w_1 and w_2 share the same parent synset, we add $(w_1,w_2,|)$ and $(w_2,w_1,|)$ to $\mathcal{L}$.
- We remove word pairs from $\mathcal{L}$ where neither word occurs in the SICK or the SNLI dataset.

2.4 Experiments

2.4.1 Direct Evaluation of Entailment Vectors

As a first step to evaluate our method, we first test whether our entailment vectors learned from labeled word pairs can indeed encode lexical entailment relations. To do this, we report the prediction accuracy on $\mathcal{L}$. Recall that $\mathcal{L}$ is derived from nouns in WordNet. In total we have 13,029 unique words and 67,935 labeled word pairs in $\mathcal{L}$. The four entailment relations are quite evenly distributed except for alternation, which has roughly twice as many word pairs as the other relations.

We compare the entailment vectors learned from the standard neural network model and from the set-theoretic model. We refer to these two as **NN** and **ST**. In addition, we also consider one baseline method, which train an SVM with RBF kernel on the word2vec embeddings of two words to predict their semantic relation.[2] We refer to this baseline as **word2vec**.

For all the methods, we take two thirds of $\mathcal{L}$ for training/validation and the remaining one third for testing. We repeat this three times and report the average accuracies. We tune the hyperparameters as follows: The dimensionality of the entailment vectors is chosen from $\{25, 50, 100, 200\}$ and the learning rate from $\{0.05, 0.1, 0.15\}$. We applied stochastic gradient descent with a mini-batch size of 5 for optimization.

We show the results in terms of accuracy and F1 in Table 2.3a. We can observe the following from the table. (1) The entailment vectors give better accuracies for predicting the entailment relations compared with using word2vec embeddings. (2) Between the standard neural network model and our set-theoretic model, the set-theoretic model achieves better performance.

Levy et al. [41] previously showed that for some models using word embedding vectors to identify hypernyms, the model is essentially memorizing "prototypical"

[2]We have also tried to use a standard neural network model instead of SVM on the word2vec embeddings, and the performance is 68.5%, which is similar to using SVM.

hypernyms. In order to check whether this happens to our model, we create a special set of labeled word pairs in which we introduce some "confusing" word pairs. We use the following rule to generate these word pairs: If we know $(\langle w_2, w_1 \rangle, \sqsubset)$ and $(\langle w_1, w_3 \rangle, |)$, then we can infer that $(\langle w_2, w_3 \rangle, |)$. We obtain 719 labeled word pairs in this way. For example, $(\langle$*staple gun, musical instrument*$\rangle, |)$ is one in this set. We denote this dataset as confusion set (CS). The accuracy and F1 scores are shown in the last two columns of Table 2.3a, respectively. We can see that the entailment vectors learned using the standard neural network models perform very poorly on this special set, suggesting that the learned entailment vectors may be memorizing the prototypical hypernyms. Using word2vec embeddings performs poorly, too. However, entailment vectors learned from the set-theoretic model perform very well, suggesting that the set-theoretic model we proposed does not simply learn "prototypical" hypernyms.

2.4.2 Evaluation on Textual Entailment

In this section, we evaluate whether the learned entailment vectors can help RTE. Here the entailment vectors are trained using the entire set of labeled word pairs derived from WordNet as we have described in Section 2.3.3. We use the SICK dataset and SNLI dataset for this experiment.

Recall that we have earlier hypothesized that when there are many word pairs in the test data that are not found in the training data, external lexical entailment knowledge about these word pairs is especially important. In order to verify this hypothesis, we also construct a different split of the data. We use the following steps to build this split:

- For every sentence pair, find all possible matching word pairs that occur in the WordNet data. E.g., we extract $\langle$someone, someone$\rangle$, $\langle$stirring, stirring$\rangle$, $\langle$chili, chili$\rangle$ and $\langle$kettle, pot$\rangle$ from the sentence pair in Table 2.1.
- For every word pair, randomly assign all sentence pairs containing such word

	Acc±SD	F1±SD	d	Acc(CS)	F1(CS)
NN	90.19±0.005	89.24±0.458	50	13.35	23.56
ST	**94.51**±0.002	**93.96**±0.106	200	**93.32**	**96.55**
word2vec	69.01±0.002	67.70±0.354	300	53.00	69.28

(a)

Dataset	SICK(ss)	SICK(ns)	SNLI(ss)	SNLI(ns)
train	4439	7163	549367	460451
dev	485	795	9842	51695
test	4906	1882	9824	61054

(b)

Model	SICK (ss)		SICK (ns)		SNLI (ss)		SNLI (ns)	
	Dev	Test	Dev	Test	Dev	Test	Dev	Test
w2v	84.2	84.1	85.4	81.8	86.5	86.2	84.7	82.8
w2v+NN	84.4	85.1*	**86.3**	84.9**	86.6	86.3	85.0	83.3**
w2v+ST	**84.6**	**85.4****	85.8	**85.1****	**86.7**	86.5	**85.3**	**83.6****
MLN-eclassif	-	85.1	-	-	-	-	-	-
DAM	-	-	-	-	-	86.3	-	-
CAFE	-	-	-	-	-	**88.5**	-	-

(c)

Table 2.3: (a) Results of direct evaluation on $\mathcal{L}$. d is the optimal dimensionality of the entailment vectors (for NN and ST) or of the word embeddings (for word2vec) based on validation. SD stands for standard deviation. (b) Numbers of sentence pairs in the SICK and the SNLI datasets. (c) Textual entailment results on the two datasets by different methods. MLN-eclassif [5] and CAFE [70] represent the state of the art on the SICK and the SNLI datasets, respectively. DAM [5] is the original decomposable attention model. The numbers shown in the table are what was reported in these dissertations. ** and * indicate statistical significance ($p \leq 0.01$) and ($p \leq 0.05$) compared with w2v by McNemar's test, respectively.

pair into either the training set or the test set. This is to ensure that the test set contains word pairs that have not appeared in the training data at all.

- Randomly split the training set into the final training set and a development set with a ratio of 9:1.

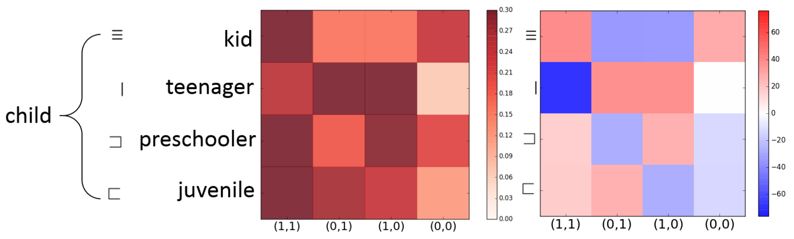
We denote the original split of the data as the "standard split" (ss) and the new split of the data as the "non-overlap split" (ns). This results in two additional specific-split datasets. Table 2.3b shows the numbers of sentence pairs of all the four datasets.

We compare the following settings of using entailment vectors in the decomposable attention model for RTE:

- w2v: This is the original decomposable attention model for RTE without using the entailment vectors. The word embeddings used are word2vec.
- w2v+NN: This setting uses the modified decomposable attention model with both word2vec word embeddings and the entailment vectors learned from the standard neural network model.
- w2v+ST: This setting uses the modified decomposable attention model with both word2vec word embeddings and the entailment vectors learned from the set-theoretic model.

The SICK dataset has a vocabulary of 2,331 words, 1,560 of which can be found in $\mathcal{L}$. For the SNLI dataset, the vocabulary contains 32,497 words, of which only 3,599 appear in $\mathcal{L}$. Following the experimental setting of the baseline [56], we fix the word embeddings. During training, we experiment with 2-layer feed-forward neural networks with 200 neurons in all sets. Learning rate is set to be 0.03 and batch size is 30. In order to better compare different models, we use the McNemar test to test the statistical significance of the performance differences between different models.

We show the experiment results in Table 2.3c. We have the following observations. (1) The result of our baseline w2v on the standard split of the SNLI dataset is almost the same as the one reported by Parikh et al. [56]. This shows that our baseline implement is as strong as the DAM model. (2) For the standard split of the datsets, both w2v+NN and w2v+ST can outperform w2v, and w2v+ST's accuracy is significantly better than w2v. In particular, our model outperforms the state-of-the-art performance on the SICK dataset. Even though the performance on SNLI is not better than the best result reported [70], where extensive syntactic features were used, we still can outperform the basic model [56] by injecting WordNet knowledge into the model. (3) For the non-overlap split, especially on the SICK dataset, using only word2vec, the RTE performance on the development set is quite high, but when it comes to the test set, the performance drops drastically. This demonstrates that



(a) (b)

Figure 2.2: (a) Visualization of the 4-dimensional $\mathbf{h}$ for four types of relations with the corresponding word pairs labeled on the left and the meaning of the 4 dimensions labeled at the bottom. (b) Visualization of the weights applied to $\mathbf{h}$ with the corresponding relations labeled on the left and the meanings of the 4 dimensions labeled at the bottom.

without observing the necessary word pairs from the training set, it is hard to make the right predictions on the test set. In contrast, using entailment vectors performs better in the non-overlap split setting. For the SNLI dataset, the improvement is not so obvious. This may be because SNLI has a low percentage of words that can be found in WordNet. (4) Comparing w2v+NN and w2v+ST, we can see that w2v+ST performs slightly better than w2v+NN most of the time. This shows the advantage of our set-theoretic model for learning the entailment vectors.

2.4.3 Further Analyses

We also conduct some further analyses on the results to better understand our method.

Analysis of the Set-Theoretic Model

Recall that for the set-theoretic model, the hidden vectors $\mathbf{h}$ as defined in Eqn. (2.2) roughly correspond to the four counters $c_{p,q}$, and these counters are essential to how the set-theoretic model works. To see whether the set-theoretic model indeed works as we have expected, we plot the values of $\mathbf{h}$ for some example word pairs. Figure 2.2a shows the $\mathbf{h}$ vectors for four word pairs from WordNet with different lexical entailment relations. Figure 2.2b shows the weights applied to $\mathbf{h}$ as defined in Eqn. (2.1) to make the final predictions of lexical entailment relations. We can see in Figure 2.2a that for the word pair $(\langle \text{child, juvenile} \rangle, \sqsubset)$, it has a large value for the counter $c_{0,1}$, and this counter has a positive weight for relation $\sqsubset$ in Figure 2.2b. On the other hand, for $\sqsupset$, the counter $c_{1,0}$ has a positive weight. Also, $|$ needs positive weights for $c_{1,0}$ and $c_{0,1}$ while $\equiv$ needs negative weights for them. All these match our expectations from Table 2.2b.

Analysis of Concrete and Abstract concepts

In Section 2.3.1, we show that our entailment vectors are mainly inspired by the set-theoretic definitions of basic semantic relations. Intuitively they work mostly for nouns presenting concrete concepts such as “animal” and “dog.” It would be interesting to see how well they work for abstract nouns such as “idea” and “proposal.” In order to check this, we conduct the following analysis.

We make use of an existing dataset[3] that contains concreteness ratings for 40,000 common English lemmas, collected using Amazon Mechanical Turk [14]. The ratings are from 0 to 5. We select words with scores 4.5 and higher to form our concrete concept list and words with scores 2.5 and lower to form another abstract concept list. This results in 596 concrete words and 250 abstract words. We then re-run the textual entailment experiments on the SICK data with the “non-overlap split” (ns). But instead of incorporating the entailment vectors of all words found in $\mathcal{L}$, we try two new settings, where we incorporate only the entailment vectors of the words found in the concrete word list or the abstract word list. We use entailment vectors learned from the set-theoretic model. We refer to these two new settings as w2v+ST(concrete) and w2v+ST(abstract). We show the performance in terms of accuracy in Table 2.4. For comparison, we also show w2v, which is our baseline that does not use any entailment vectors, and w2v+ST, which uses entailment vectors of all words. As we can see from the table, incorporating entailment vectors of concrete words works slightly better than of abstract words, but both settings perform better than the baseline w2v. This shows that even for abstract concepts, our

[3]Available at `http://crr.ugent.be/archives/1330`

	w2v	w2v+ST(concrete)	w2v+ST(abstract)	w2v+ST
SICK(ns)	81.8%	84.4%	84.0%	85.1%

Table 2.4: Evaluations of concrete concepts and abstract concepts on SICK(ns) data.

sentence pair	ground truth	w2v	w2v +NN	w2v +ST	WordNet triplet
Someone is stirring chili in a kettle Someone is stirring chili in a pot	*e*	*n*	*n*	*e*	$(\langle$*kettle, pot*$\rangle, \sqsubset)$
The black and white dog is running inside The black and white dog is running outside	*c*	*n*	*n*	*c*	$(\langle$*inside, outside*$\rangle, \mid)$
A shirtless man is playing football on a lawn A shirtless man is playing football on a field	*n*	*e*	*e*	*e*	$(\langle$*lawn, field*$\rangle, \sqsubset)$
A man is hanging up the phone A man is making a call on a cell phone	*c*	*n*	*n*	*n*	

Table 2.5: Examples of successful and erroneous predictions. "E" indicates "entailment", "n" indicates "neural" and "c" indicates "contradition".

model is still able to capture lexical-level entailment knowledge and incorporate such knowledge into the textual entailment model. Overall, incorporating entailment vectors of all words still works the best.

Case Studies

In Table 2.5 we show some successful as well as erroneous RTE predictions made by our method using the entailment vectors trained by the set-theoretic model. In the first example, since the words *kettle* and *pot* never co-occur in the training set, using only word2vec embeddings makes a wrong prediction. But injecting the entailment vectors trained using the set-theoretic model from the WordNet labeled word pairs, the knowledge that *kettle* entails *pot* is brought into the model, and therefore the w2v+ST method can correctly make a prediction. We can see that the w2v+NN method could not inject such knowledge either. The same thing happens in the next example, where WordNet provides the knowledge $(\langle$*inside, outside*$\rangle, \mid)$, which is never learned from the training set.

We also provide two negative examples where w2v+ST makes wrong predictions. In the third example, the knowledge $(\langle$*lawn, field*$\rangle, \sqsubset)$ provided by WordNet is not applicable, because here *lawn* and *field* are considered to be different. The last example shows that for paraphrases not at the lexical level, our entailment vectors cannot help.

2.5 Conclusions

In this dissertation, we propose to learn entailment vectors that encode lexical entailment knowledge from WordNet. We incorporate such entailment vectors into a decomposable attention model for RTE. Our empirical evaluation shows that the entailment vectors can indeed improve the performance of RTE when evaluated on the SICK and the SNLI datasets. In the future, we plan to study how to automatically obtain more training data for learning entailment vectors and how to encode phrase-level entailment relations in such entailment vectors. Our data and code have been made publicly available.[4]

[4]https://github.com/lanyunshi/embedding-for-textual-entailment

Part II

In Part I, we present our work on using WordNet for textual entailment. In Part II of the dissertation, we turn to another task that uses knowledge bases. This is the task of knowledge base question answering. We first review the related literature below and then describe our detailed work in the following chapters.

KBQA has been studied extensively in recent years. We can summarize the solution of KBQA into two main categories. One category is a kind of semantic parsing-based methods [90, 15, 6, 35, 25, 86, 42]. The idea is to transform the natural language question into a specific query-language form so that we can use it to directly retrieve answers from a KB. However, this category of methods faces the challenge of building accurate and domain-specific parsers. Another category of methods matches questions with candidate answers via neural network models [10, 12, 11, 22, 31, 29, 88]. Different features and neural networks are proposed to improve the accuracy of matching. [10] represented a question and a knowledge base constitute as two low-dimensional vectors and computed their similarity, but their method ignored word order information and importance of different words. [88] represented questions and relations of candidate answers via different levels of abstraction and sorted relations via the neural network in their staged framework. Their method makes full use of the question and relations but overlooks other important information in the KB. [29] collected multiple aspects of the information about a candidate answer as the representation of the candidate. Then a cross-attention mechanism was proposed to match it with the question. In addition they used the TransE method to train representations for candidate answers using global KB knowledge. Our work in Chapter 3 follows the second line of work. We try to adopt a more expressive matching neural network and leverage a more informative feature to do the matching. It's worth noting that the work [29] also incorporate some additional knowledge, but there are still some differences: Their candidate answers are not represented as sequences whereas we use sequence representation for candidates. They use TransE to incorporate additional knowledge about candidates, but TransE requires the entire KB to train. In contrast, we only need to use the local contexts of candidates. Sun et al. [68] adapted graph convolutional network (GCN) to encode the KB knowledge in KBQA task. So that each entity could be represented by its neighborhood via attention mechanism. But the difference is that the information of GCN model propagate between any two entities for ranking the entity while our method is one-way representation from relations to an entity for describing the entity. Some other work like [83] and [78] incorporated manually designed rules or Wikipedia as external knowledge to help KBQA. In the experiment in Chapter 3, we do not make use of any external knowledge. Therefore, we do not compare with their reported performance.

Our work in Chapter 4 studies the overall architecture of KBQA. A general solution to KBQA is a pipeline approach, where the question is first linked to some topic entities in the KB by an existing entity linking tool [82, 11, 21, 29] and then relation paths connected to the topic entities in the KB are retrieved and ranked by various ranking models. As we mentioned, these methods include semantic parsing based models [6, 83, 85] and embedding based models [10, 11, 21, 68]. The top-ranked relation path will then be used to answer the question. Although these methods have worked well on KBQA, there are limitations of this pipeline approach using an external entity linking tool [66, 36, 45], as we have pointed out in Section 1. There has been some work attempting to address the limitations pointed out. Yu et al. [88] re-rank the topic entities using their associated relations. Xu et al. [78] jointly train entity linking and path ranking through entity descriptions. However, these studies still only consider named entities as topic entities rather than linking a question to other kinds of KB units. In contrast, we consider a wide range of KB units during our topic unit linking step. Zhang et al. [92] integrate entity linking and relation path ranking into an end-to-end model. However, their topic entity linking module considers all candidate topic entities without using a scoring function to rank and filter them, and as a result, it is hard to scale up their method to large knowledge bases. In contrast, our work in Chapter 4 takes a topic unit generation-and-scoring approach, using heuristics for generating topic units and a neural network model for fine-grained scoring. Similar to [92], We also use reinforcement learning to jointly train the entire system.

Along the line of KBQA, most previous work focuses on single-relation questions. For multi-hop KBQA, there has been only limited work. We then investigated the multi-hop KBQA task in Chapter 5. [76] applied Memory Network to KBQA, where relation triplets are saved in memory and queries are updated at every hop by interacting with the memory. The Key-Value Memory Network method [50] improves Memory Network by dividing the memory into key and value parts. [92] proposed a variational reasoning network that integrates entity linking and relation prediction together and performs reasoning by traversing within the KB. [94] proposed an interpretable reasoning network, which performs relation matching hop-by-hop and has shown good performance on some datasets. However, as we discussed earlier, these methods have several limitations. We propose a new method in our third work in Chapter 5 to address these limitations and we show that our method can outperform these previous methods. A recently-published work [18] proposed a general framework to address the multi-hop question answering task, which achieved outstanding results for evaluation. The difference between their method and ours is that they re-visit the ranked paths during the expansion of the relation paths while we propose a more complex mechanism to remember the visited paths.

Chapter 3

Knowledge Base Question Answering with a Matching-Aggregation Model and Question-Specific Contextual Relations

3.1 Introduction

With the development of large-scale knowledge bases such as Freebase [9] , DBpedia [1] and YAGO [67], Knowledge Base Question Answering (KBQA) has become an important task and gained much attention in recent years [10, 12, 93, 88, 29]. KBQA aims to automatically find answers to factoid questions from a Knowledge Base (KB), where answers are usually entities in the KB. Figure 3.1 shows a small subset of a KB and an example question that can be answered from the KB.

Early work on KBQA often uses semantic parsing to transform a question into a KB-specific structure that can be used to directly query or match the KB [90, 72, 6, 15, 84, 83, 42]. However, this approach depends heavily on a suitable and accurate semantic parser, which may not be easy to build. More recently, a number of neural network-based methods have been proposed for KBQA and achieved good results

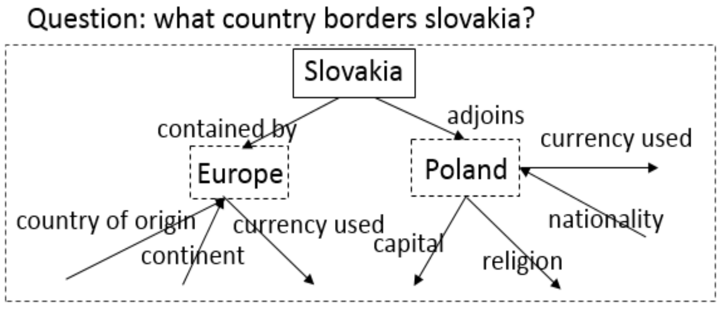


Figure 3.1: An example question and a subgraph of the KB. The correct answer to the question is *Poland*. The entity shown in the solid rectangle is a *topic entity*, and the two entities shown in dashed rectangles are *candidate answer entities*.

on benchmark datasets [22, 29, 88].

Typically, these methods start by identifying entities mentioned in the question, which are referred to as *topic entities*. From these topic entities and by following the relation paths in the knowledge graph, candidate answer entities can be located, which are usually one or two-hops away from a topic entity. Neural network models are then used to encode both questions and candidate answers as vectors, which are then matched against each other for candidate ranking. Differences between the various models proposed lie in the types of KB information they use to represent candidate answers as well as the way they encode and match.

Although existing neural network-based methods have explored various candidate answer representations and matching models, there are at least two limitations of existing work. The first is about the matching model. KBQA can be regarded as a sequence matching problem where one sequence is the question and the other sequence is the relation path in the KB linking a topic entity to a candidate answer entity. For natural language sequence matching, several previous studies have shown that a "matching-aggregation" framework is preferred [74, 75, 56], in which two sequences are matched at word-level and the matching results are aggregated for a final decision. However, existing matching models for KBQA are not based on this "matching-aggregation" framework, and thus may not achieve optimal matching results between questions and candidate answers.

Second, to represent candidate answers, existing methods usually consider only the entities and relations along the path from a topic entity to a candidate answer entity. A recent work [89] proposed to directly leverage the type mentions of the candidate for matching, such as “location”. However, we think other relations linked to a candidate answer may also contain more expressive information about the candidate and should be considered. Take the example shown in Figure 3.1. We can see that the relations “capital” and “nationality” linked to the candidate “Poland” strongly indicate that this candidate is a country. If this information is considered during matching, we can increase the chance of this candidate being ranked high.

In this chapter of the dissertation, we address the two limitations above by proposing two ideas for KBQA. First, we apply a “matching-aggregation” model to measure the similarity between a question and a candidate answer, which allows us to exploit word-level interactions through bi-directional attention mechanisms. Second, we incorporate additional relations connected to a candidate to enhance its representation, and we use attention mechanism to weigh these relations based on their relevance to the question. We evaluate our proposed method on two data sets: WebQuestions [6] and SimpleQuestions [11]. The empirical results verified our hypotheses that a “matching-aggregation” framework works better for finding correct answers in KBQA, and the additional relations incorporated into the candidate representations can further improve KBQA performance. We also find that our overall method can outperform the reported state-of-the-art performance on both WebQuesions and SimpleQuestions datasets by 4.2 percentage points and 2.2 percentage points, respectively.[1]

The contributions of our work are two-fold: (1) We demonstrate that a “matching-aggregation” framework for sequence matching works significantly better than a standard sequence matching model for KBQA. (2) We propose to use additional relations connected to candidate answers to help candidate ranking, which can also significantly improve KBQA performance.

[1]For fair comparison, we do not consider models using external resources such as Wikipedia.

3.2 Method

3.2.1 Task Definition and Setup

We first formally define the task below. We assume that there is a KB from which questions are to be answered. The KB contains a set of entities $\mathcal{E}$, where each entity $e \in \mathcal{E}$ has a textual representation, which is a sequence of words. For example, (*isthmus*,*of*,*panama*) is the textual representation of the entity *Isthmus of Panama*. The KB also has a set of relations defined, denoted by $\mathcal{R}$, where each relation $r \in \mathcal{R}$ also has a textual representation in the form of a word sequence (e.g., (*contained*,*by*)). We assume that all relations in the KB are directed, binary relations.[2] Facts in this KB are represented as triplets. For example, (e,r,e') indicates that the relation $r \in \mathcal{R}$ holds between the entities $e \in \mathcal{E}$ and $e' \in \mathcal{E}$, where e is called the *head entity* and e' the *tail entity*. The KB can also be represented as a graph, as we can see in Figure 3.1.

A question is represented as a sequence of words $Q=(q_1,q_2,\ldots,q_m)$ (e.g., (*what*,*country*,*borders*,*slovakia*)). Our goal is to return an entity from $\mathcal{E}$ as the answer to a given question. We assume that there is a set of question-answer pairs used as training data.

3.2.2 Method Overview

Before presenting the details of our method, we first give an overview. Similar to most previous work, our method begins by identifying *topic entities*, which are entities mentioned in a given question. We then use these topic entities to identify *candidate answer entities* (or *candidates* for short). In this work, candidates are those entities that are either directly linked to a topic entity through a single relation or two-hop away from a topic entity through two relations in the KB. In the example shown in Figure 3.1, *Slovakia* is a topic entity, and two candidates can be

[2]For n-ary relations, we convert them to binary relations following the practice by [31].

candidate	base candidate sequence	additional relations
Europe	(*slovakia*,*contained*,*by*)	{*#continent*$^{-1}$, *#currency_used*, *#country_of_origin*$^{-1}$}
Poland	(*slovakia*,*adjoins*),	{*#nationality*$^{-1}$, *#religion*, *#capital*, *#currency_used*}

Table 3.1: Two candidates and their candidate sequences for the question "*what country borders slovakia*." Note that X^{-1} indicates the inverse relation of X.

derived from the topic entity: *Europe* and *Poland*. For each candidate, we use the entities and relations along the path from the original topic entity to the candidate to construct a sequence, which we refer to as a *candidate sequence*. Details of candidate sequences will be presented in Section 3.2.3 and Section 3.2.4. Finally, using a neural network-based sequence matching model, we match all candidate sequences with the question sequence in order to rank the candidates and select the top one as the answer.

Although the overall framework of our method is similar to previous work, we propose two novel ideas to address the limitations we pointed out earlier in order to improve KBQA performance. (1) To match the question sequence with a candidate sequence, we adopt a "matching-aggregation" framework, which has been shown to be more effective than a "Siamese" matching model for various NLP tasks [74, 75, 56, 53]. Note that although this "matching-aggregation" framework is not new, to the best of our knowledge, it has not been applied to KBQA. (2) Based on our observation that additional relations connected to the candidates are potentially useful, we include them in the candidate representations. We also note that these relations are not equally important and therefore we propose to use an attention mechanism to carefully weigh these relations in order to optimize their effect.

We now present our method in detail.

3.2.3 Base Candidate Sequences

In order to identify a set of entities from the KB as candidate answers, we first identify a set of *topic entities* from the given question. For example, given the question "*where is isthmus of panama located*," we would identify *Isthmus of Panama* and *Panama* as topic entities. Note that this step follows the practice of several previous studies [83, 29, 88]. We use external tools to identify the topic entities. Let us use $\mathcal{E}_Q^t \subset \mathcal{E}$ to denote the set of topic entities found in question Q.

Next, for each topic entity $e^t \in \mathcal{E}_Q^t$, by following its connections in the KB, we can identify all the entities that are either one-hop or two-hop away from e^t. We combine all these entities that are one or two-hop away from any topic entity and consider them to be our candidate answer entities. Let us use $\mathcal{E}_Q^c \subset \mathcal{E}$ to refer to this candidate set.

We first introduce our *base candidate sequences*, which do not include the additional relations. To construct the *base candidate sequence* for a candidate $e^c \in \mathcal{E}_Q^c$, we use the entities and relations along the path connecting this candidate to the corresponding topic entity. Recall that each entity or relation has a textual representation in the KB. We concatenate the word sequences representing the entities and relations along the path to form the base candidate sequence. Note that we exclude the candidate entity itself in the base candidate sequence representation. This is because eventually we will match the candidate sequence with the question sequence, but we do not expect the candidate itself to appear in the question. For example, the candidate *Poland* does not appear in the question *"what country borders slovakia"* although this candidate is the correct answer.

To illustrate base candidate sequences, we show two candidates and their corresponding base candidate sequences in Table 3.1, corresponding to the example shown in Figure 3.1.

3.2.4 Enhanced Candidate Sequences

To enhance the base candidate sequence for a candidate e^c, we further look for other relations that are linked to e^c, where e^c could be either a head entity or a tail entity. Following Hao et al. [29]'s work, we have tried to involve candidate answer entity itself into the candidate sequence. However, based on our preliminary experiments, we find that indeed such additional information does not improve the performance. Let $\mathcal{R}_{e^c} \subset \mathcal{R}$ denote the set of relations connected to e^c, excluding those that link e^c to a topic entity. This set of additional relations $\mathcal{R}_{e^c}$ will be considered an additional component in the enhanced candidate sequence. For example, for the candidate *Poland*, *nationality* would be one of the additional relations, which could potentially help better match this candidate. See Table 3.1 for the additional relations of the two candidates in our example.

3.2.5 Sequence Matching

We are now ready to use a sequence matching model to measure how close a candidate sequence is to a question sequence. Inspired by some recent work [56, 74], we apply a "matching-aggregation" sequence matching model here.

First of all, for the question sequence and the base candidate sequence, we associate each word with a word embedding vector (which will be initialized using existing word embeddings but updated during training). Then for the set of additional relations in the enhanced candidate sequence, we associate each additional relation with a relation embedding vector (which will be randomly initialized and updated during training).

Let $\mathbf{Q} = (\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m)$ denote the sequence of word embeddings of the question. Let $\mathbf{C} = (\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_n)$ denote the sequence of embedding vectors of the enhanced candidate sequence for candidate e^c. Here $(\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{n-1})$ are the word embeddings of the words in the base candidate sequence for e^c, and the last embedding $\mathbf{c}_n$ is defined as a combination of the relation embeddings of the relations inside $\mathcal{R}_{e^c}$, i.e., the additional relations linked to e^c. We will explain how $\mathbf{c}_n$ is derived from the relations in $\mathcal{R}_{e^c}$ later.

Given the two sequences $\mathbf{Q}$ and $\mathbf{C}$, we try to derive a matching score between them as follows. First, we try to match $\mathbf{q}_i$ with $\mathbf{c}_j$ as follows:

$$e_{ij}{=}F(\mathbf{q}_i)^T F(\mathbf{c}_j),$$

where $F(\cdot)$ is a single non-linear layer with ReLU as its activation function.

We then use e_{ij} defined above to derive normalized attention weights in both directions and obtain the following weighted versions of the question (or candidate) to match each word in the candidate (or question) sequence:

$$\tilde{\mathbf{q}}_j{=}\sum_{i=1}^{m}\frac{\exp(e_{ij})}{\sum_{i'=1}^{m}\exp(e_{i'j})}{\cdot}\mathbf{q}_i,$$
$$\tilde{\mathbf{c}}_i{=}\sum_{j=1}^{n}\frac{\exp(e_{ij})}{\sum_{j'=1}^{n}\exp(e_{ij'})}{\cdot}\mathbf{c}_j.$$

We can see that here $\tilde{\mathbf{q}}_j$ is a weighted sum of all the $\mathbf{q}_i$ in the question sequence in order to match $\mathbf{c}_j$ in the candidate sequence. It follows the standard attention mechanism in most previous work. The same idea applies to $\tilde{\mathbf{c}}_i$.

Next, we match $\mathbf{q}_i$ with $\tilde{\mathbf{c}}_i$ and $\mathbf{c}_j$ with $\tilde{\mathbf{q}}_j$ by defining the following two vectors:

$$\mathbf{v}_{1,i}{=}G\left(\begin{bmatrix}\mathbf{q}_i\odot\tilde{\mathbf{c}}_i\\(\mathbf{q}_i-\tilde{\mathbf{c}}_i)\odot(\mathbf{q}_i-\tilde{\mathbf{c}}_i)\end{bmatrix}\right),$$
$$\mathbf{v}_{2,j}{=}G\left(\begin{bmatrix}\mathbf{c}_j\odot\tilde{\mathbf{q}}_j\\(\mathbf{c}_j-\tilde{\mathbf{q}}_j)\odot(\mathbf{c}_j-\tilde{\mathbf{q}}_j)\end{bmatrix}\right),$$

where $\odot$ denotes element-wise multiplication and $G(\cdot)$ is another feed-forward neural network with ReLU activation. Note that our design of these two vectors are inspired by some recent work on sequence matching [74].

Next, we aggregate the sequences of $\mathbf{v}_{1,i}$ and of $\mathbf{v}_{2,j}$ using LSTM and then

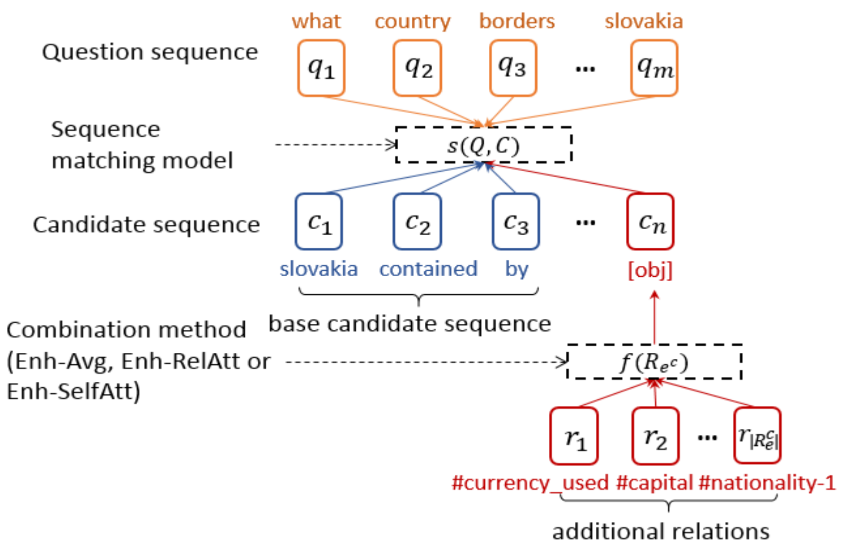


Figure 3.2: An illustration of our model.

extract two values from the resulting vectors through max pooling:

$$\tilde{\mathbf{V}}_1 = \text{LSTM}([\mathbf{v}_{1,1}, \mathbf{v}_{1,2}, \ldots, \mathbf{v}_{1,m}]), \qquad \tilde{\mathbf{v}}_1 = \max_i \tilde{\mathbf{V}}_{1,i},$$

$$\tilde{\mathbf{V}}_2 = \text{LSTM}([\mathbf{v}_{2,1}, \mathbf{v}_{2,2}, \ldots, \mathbf{v}_{2,n}]), \qquad \tilde{\mathbf{v}}_2 = \max_j \tilde{\mathbf{V}}_{2,j}.$$

Finally, we concatenate and feed $\tilde{\mathbf{v}}_1$ and $\tilde{\mathbf{v}}_2$ to H, which is a feed forward network followed by a linear layer. It gives us the matching score between question sequence $\mathbf{Q}$ and candidate sequence $\mathbf{C}$:

$$s(\mathbf{Q}, \mathbf{C}) = H([\tilde{\mathbf{v}}_1; \tilde{\mathbf{v}}_2]).$$

Using a softmax layer over the matching scores of all candidates, we can then derive a distribution over the candidates.

During the training stage, we use the KL divergence between the true distribution and predicted distribution as the objective function to learn the various model parameters. For prediction, we select the candidate with the highest probability as the predicted answer.

3.2.6 Combining Additional Relations with Attention

We now describe how $\mathbf{c}_n$ is derived from the additional relations $\mathcal{R}_{e^c}$ for candidate e^c.

A naive way is to take the average of all the relation embeddings, which we refer to as **Avg**:

$$\mathbf{c}_n = \frac{1}{|\mathcal{R}_{e^c}|} \sum_{r \in \mathcal{R}_{e^c}} \mathbf{r},$$

where $\mathbf{r}$ is the embedding vector of relation r.

However, this naive method has its weakness because not all additional relations are equally relevant to the question. To better capture the relevant additional relations, we use attentions to weigh the different additional relations.

We first encode the question sequence $\mathbf{Q}$ into a single vector $\bar{\mathbf{q}}$ by taking the average.

$$\bar{\mathbf{q}} = \frac{1}{m} \sum_{i=1}^{m} \mathbf{q}_i.$$

After that we apply an attention network to decide which relation is important for the question. We define β_r as follows:

$$\beta_r = \frac{\exp(\mathbf{w}^T[\mathbf{r};\bar{\mathbf{q}}]+b)}{\sum_{r' \in \mathcal{R}_{e^c}} \exp(\mathbf{w}^T[\mathbf{r}';\bar{\mathbf{q}}]+b)}, \tag{3.1}$$

where $\mathbf{w}$ and b_r are parameters to be learned, and $\mathbf{r}$ is the embedding for relation r. Then $\mathbf{c}_n$ is obtained as follows:

$$\mathbf{c}_n = \sum_{r \in \mathcal{R}_{e^c}} \beta_r \mathbf{r}.$$

We refer to this combination method as **RelAtt**.

Still, for a question $\mathbf{Q}$, oftentimes only some aspects of the question are more important, such as "country" in the example question we have seen. So we also explore a self-attention mechanism on the question to decide which parts of the question should be highlighted to match a candidate. We first use LSTM to transform a question sequence into a single vector:

$$\tilde{\mathbf{Q}} = \text{LSTM}([\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m]), \qquad \tilde{\mathbf{q}} = \tilde{\mathbf{Q}}_m.$$

Next, we define

$$\alpha_i = \frac{\exp(\mathbf{u}^T[\tilde{\mathbf{q}}; \mathbf{q}_i] + d)}{\sum_{i'=1}^{m} \exp(\mathbf{u}^T[\tilde{\mathbf{q}}; \mathbf{q}_{i'}] + d)},$$

where $\mathbf{u}$ and d are parameters to be learned.

Then we define

$$\bar{\mathbf{q}}' = \sum_{i=1}^{m} \alpha_i \mathbf{q}_i.$$

Then we can use $\bar{\mathbf{q}}'$ instead of $\bar{\mathbf{q}}$ in Equation (3.1) to obtain $\mathbf{c}_n$. We denote this method as **SelfAtt**.

3.2.7 Implementation Details

It is worth noting that some questions may give strict constraints of the answer type. For example, for the question *"what state is Harvard College located"*, the candidate answers include *Cambridge*, *United States of America* and *Massachusetts*. If we directly have the entity type information of these candidates, we can easily find that *Massachusetts* is the best answer. Although our method using additional relations could possibly also encode such knowledge, we expect that using explicit entity type or entity description would possibly be supplementary because they can encode more find-grained entity type information. Thus, we include a post-processing step with the following heuristic. If we find some exact word match between the question and either a candidate entity's textual description or the entity type of the candidate, we adjust the probability of the candidate by the following formula:

$$p(e^c)' = \gamma + (1 - \gamma) \times p(e^c), \tag{3.2}$$

where $p(e^c)$ is the probability for candidate e^c as computed by the sequence matching model, and γ is a hyper-parameter manually set.

It is also possible that sometimes there may be many candidates sharing the same base candidate sequence. This is because there are often one-to-many relations in a KB. To reduce the computational costs, instead of treating these as different candidate sequences, we merge these candidates as well as their additional relations and construct a single candidate sequence for them. We use the heuristic explained above to further rank them.

Note that these implementation details are applied to all versions of our model that are being compared in Section 3.3.

3.3 Experiments

3.3.1 Setup

We evaluate our proposed method on two commonly used benchmark datasets: WebQuestions and SimpleQuestions.

WebQuestions[3]: This dataset was introduced by Jonathan et al. [6]. The dataset contains 5,810 question-answer pairs with 3,778 training pairs and 2,032 test pairs.

We randomly split the training data into 3,000 training pairs and 778 development pairs. In order to obtain topic entities, we use the entity linking output generated by YodaQA[4]. The KB we use is the latest dump of Freebase[5] and we process it the same way as [31].

Due to multiple correct answers for each question are annotated as the ground truth, our method also return multiple answers based on a tuned threshold. We evaluate our method using the official evaluation script provided by Jonathan et al. [6]. The standard valuation metric is average F1 over all test questions.

[3]https://nlp.stanford.edu/software/sempre/
[4]https://github.com/brmson/dataset-factoid-webquestions
[5]https://developers.google.com/freebase/

SimpleQuestions[6]: The SimpleQuestions dataset was introduced by Bordes et al. [11]. It contains 108,442 question-answer pairs, with 75,910, 10,845 and 21,687 pairs for training, development and testing, respectively. In order to make fair comparison with previous work, we use FB2M as our KB, which is a subset of Freebase that consists of 2M entities and 6K relations. For topic entities, we start from the entity linking results from Yu et al. [88][7]. For this dataset, the standard evaluation metric is accuracy, which means we count one prediction as correct if our topic entity and relation match the ground truth.

For both datasets, we initialize our word vectors with 300-dimensional pre-trained word embeddings [57]. Adagrad [24] algorithm is employed to optimize our objective function. We tune the hyper-parameters on the development data in the following way: (1) The size of hidden states is chosen from $\{50,100,150,200\}$. (2) Dropout ratio is chosen from $\{0,0.1,0.2,0.3,0.4\}$. (3) hyper-parameter γ for post-processing is chosen from $\{0.1,0.2,0.3,0.4,0.5\}$.

Through the empirical evaluation we aim to test (1) whether the "matching-aggregation" model works better than a standard sequence matching model for KBQA, and (2) whether our enhanced candidate sequences with the additional relations are better than the base candidate sequences. Therefore, we compare the following methods:

Public Baselines: We list the performance of previous works building on end-to-end neural networks [10, 22, 11, 29, 88] or semantic parsers [3, 79, 7, 80] to solve WebQestions or SimpleQuestions.

My Baseline: This is a baseline method implemented by ourselves, where we use the base candidate sequences and a standard sequence matching model that does not follow the "matching-aggregation" framework. Specifically, we use a BiLSTM model to process both the question sequence and the candidate sequences first. We then use max pooling to combine all the hidden states of a question (or a candi-

[6]https://research.fb.com/downloads/babi/

[7]https://github.com/Gorov/SimpleQuestions-EntityLinking

date sequence) into a single vector. These vectors are then used to compute cosine similarities between all candidate sequences and questions to rank the candidates.

Match-Aggr: This is our method that uses the base candidate sequences together with the "matching-aggregation" framework for sequence matching, as presented in Section 3.2.5.

Enh-Avg: This is our method using the enhanced candidate sequence with the Avg method to combine the additional relations.

Enh-RelAtt: This is our method using the enhanced candidate sequence with the RelAtt method to combine the additional relations.

Enh-SelfAtt: This is our method using the enhanced candidate sequence with the SelfAtt method to combine the additional relations.

For the sake of completeness, we also list the best reported performance on these datasets. However, it is important to note that the best performing systems [83, 78] make use of external resources such as Wikipedia. Because we do not make use of such external resources, it is not fair for us to compare our results with these state-of-the-art results.

3.3.2 Results

Our results are shown in Table 3.2. As we can see from the table, Match-Aggr method beats Baseline model with a vast margin and it can already reach the state-of-the-art performance on both datasets. Next, we can see that after we use enhanced candidate sequences with the additional relations, even with the Avg combination method, the performance can be significantly improved. With the attention mechanisms, the performance can be further improved significantly, and specifically, SelfAtt performs better than RelAtt. Overall, with our complete method, we can improve the performance of KBQA by around 4 and 2 percentage points for WebQuestions and SimpleQuestions datasets respectively.

Method	**WQ** Avg F1	**SQ** Avg acc
Bao et al. [3]	37.5	-
Xu et al. [79]	39.1	-
Bordes, Chopra and Weston [10]	39.2	-
Berant and Liang [7]	39.9	-
Yang et al. [80]	41.3	-
Dong et al. [22]	40.8	-
Bordes et al. [11]	42.2	63.9
Yin et al. [87]	-	76.4
Hao et al. [29]	42.9	-
Yu et al.[88]	-	78.7
My Baseline	39.5	74.1
Match-Aggr	42.9	79.2
Enh-Avg	43.8^{*}	80.3^{*}
Enh-RelAtt	$45.2^{*\dagger}$	$80.7^{*\dagger}$
Enh-SelfAtt	$\mathbf{47.1}^{*\dagger}$	$\mathbf{80.9}^{*\dagger}$
Yih et al.[83]	52.5	-
Xu et al. [78]	53.3	-

Table 3.2: Experiment results. The top section contains previously reported performance on the two datasets. The middle section contains our results. The bottom section serves as a reference point to show the state of the art. However, the two studies in the bottom section used external resources such as Wikipedia. $*$ and $\dagger$ indicate that the result is statistically significantly better than Match-Aggr and Enh-Avg, respectively.

3.3.3 Further Analyses

In this section, we perform some further analyses to better understand our model.

Learned Embeddings of Additional Relations

To check whether the additional relations can indeed encode useful knowledge about the candidates, we extract the learned relation embeddings of some of the additional relations and map them to a 2-dimensional space. We show these relations in Figure 3.3. We can see that indeed relations that are close to each other tend to be associated with the same type of entities. For example, *country* and *gender* are close to each other in Figure 3.3, probably because both these two relations connect to entities which are people. We can also see that *opening_date* and *date_of_birth* are also close to each other, probably because they both connect to entities which are dates.

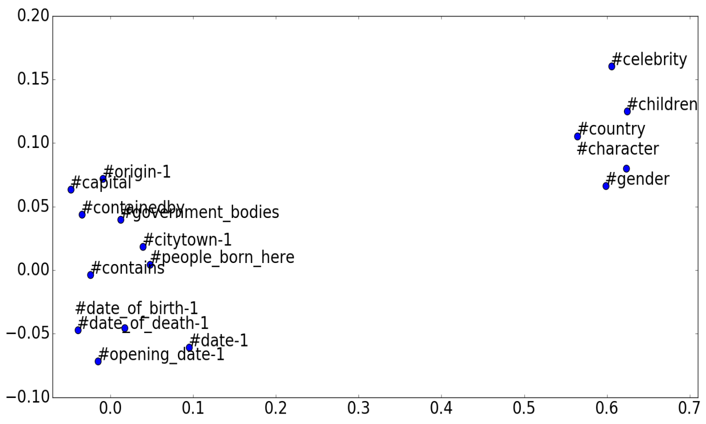
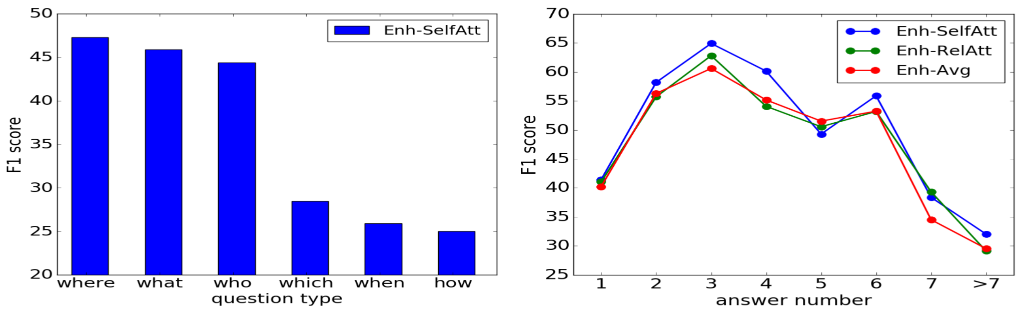


Figure 3.3: Learned relation embeddings in 2-D space.



(a) (b)

Figure 3.4: (a) F1 scores over different question types on WQ. (b) F1 scores over questions with different numbers of answers on WQ.

Performance Breakdown

To see if our method works better for some types of questions and worse for others, we group the questions in two ways.

First, we group the questions based on the answer type. This can be done by looking at the first word of a question, such as “when,”, “where.” We show the performance of different types of questions on the WebQuestions dataset in Figure 3.4a. We can see that “how” and “when” questions are harder to answer than other question types.

Because some questions have multiple answers, we also group the questions based on how many answers they have. We show the performance vs. the numbers of answers on the WebQuestions dataset in Figure 3.4b. We can see when the number of answers goes up, the performance drops. It is interesting to see that the best performance is achieved when the question has three answers. This is probably because when there are more than one answers, if we can capture one of them, we can already gain some points, but if a question's answer is unique, it is hard to rank the correct answer at the top.

Error Analysis

We also conduct some error analysis. We sample 100 imperfectly answered questions from the WebQuestions dataset randomly. We then examine them to identify the reasons for the mistakes. The following categories of errors are identified:
Ground truth incompletion (29%): There are many sampled questions whose ground truth answers are not complete. For example, for the question *"what team is Kris Humphries play for,"* besides the answer *Brooklyn Nets*, we find that there are other correct answers from the KB such as *Washington Wizards* and *Toronto Raptors*.
Question ambiguity (20%): This category contains questions which have ambiguous descriptions. As a result, multiple potential relations may match the questions correctly. For example, for the question *"who was Juan Ponce de Leon family,"* the predicted answer is *Barbara Ryan* through the *parents* relation, while the ground truth is *Elizabeth Ryan* through the *children* relation.
Complex questions (18%) : This type of errors occurs when some inference is needed to answer the question. For example, for the question *"who rules Denmark right now,"*, one needs to know the current time and compare it with the time associated with a relevant relation (e.g., *appointed_by*) in order to find the correct answer. There are also questions containing qualifiers such as *first*, *last time* and *after*, which make the questions harder to answer. For this type of questions, our method is not able to handle them.

3.4 Conclusions

In this chapter of the dissertation, we proposed a sequence matching-based solution to KBQA. We constructed candidate sequences using entities and relations linking candidate answers to a question. Furthermore, we proposed to include additional relations connected to a candidate to further enhance its representation. Our experiment results showed that our method could outperform the current state of the art for two commonly used benchmark KBQA datasets.

Chapter 4

Knowledge Base Question Answering with Topic Units

4.1 Introduction

KBQA attracts increasing attention in recent years due to its wide usage such as in search engines and decision support systems [15, 35, 10, 22, 83, 78, 88, 58]. Most existing methods for KBQA use a pipelined approach: First, given a question q, an *entity linking* step is used to find KB entities mentioned in q. These entities are often referred to as *topic entities*. Next, relations or relation paths in the KB linked to the topic entities are ranked such that the best relation or relation path matching q is selected as the one that leads to the answer entities. For example, given the question Q1 shown in Figure 4.1, an entity linking tool may link the phrase "Morgan Freeman" in the question to the entity "Morgan Freeman" in the KB. Then starting from this topic entity, a number of different relation paths are considered such as *(Morgan Freeman, education)*, *(Morgan Freeman, place of birth)* and *(Morgan Freeman, place of birth, contained by)*. Ideally, we want the path *(Morgan Freeman, education)* to be ranked the first with respect to the question so that the correct answer at the end of this path can be extracted.

Although a plethora of methods has been proposed for KBQA, most work fo-

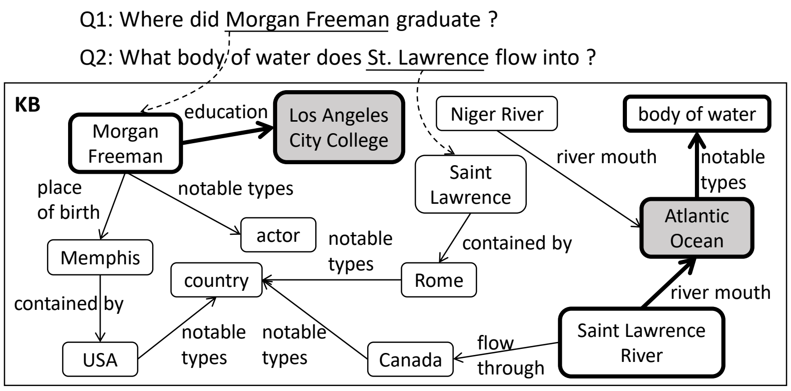


Figure 4.1: Two example questions and how they can be answered by a KB. The questions are linked to *topic entities* by imaginary lines. The shaded entities are the correct answers to the questions. The paths in bold are correct relation paths towards the questions.

cuses on the relation path ranking step. For entity linking, many methods rely entirely on existing entity linking tools [78, 21, 29], which generally use traditional rule-based methods to perform named entity recognition and linking. There are at least two limitations with this approach. First, oftentimes an entity mention in a question is ambiguous and an entity linking tool may not link it to the correct entity in the KB. Take the question Q2 in Figure 4.1 for example. An entity linking tool is more likely to mistakenly link the entity mention "St. Lawrence" to the entity "Saint Lawrence" in the KB, which is far away from the correct answer entity "Atlantic Ocean." On the other hand, the question in Figure 4.1 shows that words that are not part of a named entity, such as "body", "water" and "flow", can also be linked to relevant entities and relations in the KB such as "body of water" and "flow through" that can help find the correct answer entity. The second limitation with a pipeline approach is that the entity linking step cannot be trained using the final KBQA results. Again, let us look at the Q2 in Figure 4.1. If both "Saint Lawrence" and "Saint Lawrence River" are recognized as topic entities, an entity linking module developed outside of the KBQA system would not be able to know which one is more relevant to the question. However, if we could train a topic entity ranking function using the ground truth answer to the question, we may learn that with "water" and "flow" appearing in the question, "Saint Lawrence River" should be ranked higher than “Saint Lawrence” as a topic entity.

In this chapter of the dissertation, we address the two limitations above by replacing the standard topic entity linking module with a novel *topic unit generation-and-scoring* module. Our topic units include not only named entities but also other KB units such as entities containing common nouns (e.g., “body of water”) and relation types (e.g., “flow through,” “river mouth”). By flexibly considering a wide range of topic units, we can increase the chance of the correct answer being connected to one of the topic units. However, we do not want to consider too many topic units for the subsequent relation path ranking step as this would incur high computational costs. We therefore propose to identify topic units in two steps: a generation step and a scoring step. First, in a topic unit generation step, we use heuristics with low computational costs (e.g., n-gram matching with an inverted index) to identify an initial set of topic units that has a high coverage. Subsequently, in a topic unit scoring step, we use a neural network-based scoring function to rank the initial topic units and select a small number of them that are highly relevant to the question. We then only consider relation paths derived from this small set of topic units. Our method is trained in an end-to-end manner using reinforcement learning such that the ranking function in the topic unit scoring step can be learned without knowing the ground truth topic units.

We evaluate our method on three benchmark datasets: WebQuestionsSP, ComplexWebQuestions and SimpleQuestions. We find that our method can clearly outperform the state of the art on two datasets, especially on ComplexWebQuestions where the improvement is substantial. It also performs competitively on the third dataset. Further analyses also show that considering a wide range of topic units is crucial to the performance improvement.

4.2 Method

4.2.1 Task Setup

We first formally define the KBQA task. A KB (knowledge base) consists of a set of entities $\mathcal{E}$ (e.g., *Morgan Freeman*), a set of relation types[1] $\mathcal{R}$ (e.g., *place of birth*) and a set of relation triplets (h,r,t) (e.g., (*Morgan Freeman*,*place of birth*,*Memphis*), where $h \in \mathcal{E}$ is the head entity, $t \in \mathcal{E}$ is the tail entity, and $r \in \mathcal{R}$ is a directed relation between h and t. The triplets (h,r,t) are facts or knowledge contained in the KB. A KB can also be seen as a *knowledge graph* whose nodes are the entities and edges are the relations. We assume that each entity or relation type has a textual description, which is a sequence of words.

We define *KB units* (denoted as $\mathcal{U}$) to be the union of the entities and the relation types, i.e., $\mathcal{U} = \mathcal{E} \cup \mathcal{R}$. Given a question q, the task of KBQA (knowledge base question answering) is to find a set of entities in $\mathcal{E}$ that are answers to q. It is assumed that a set of questions together with their correct answers in the KB is given for training.

4.2.2 Method Overview

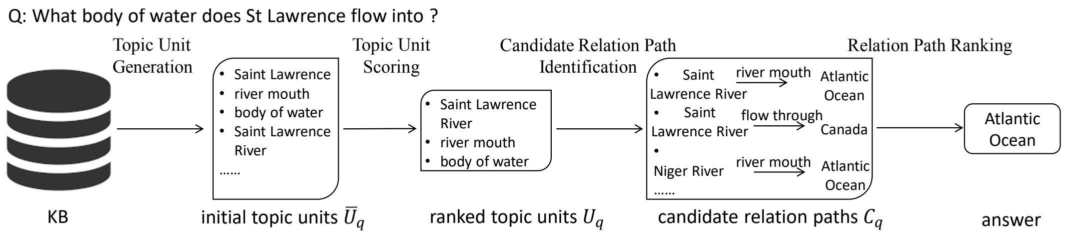


Figure 4.2: An overview of the various steps of our method.

Like many existing methods for KBQA, our method follows the general approach of first linking the words in the question to some parts of the KB as starting points for search and then following the edges of the knowledge graph to look for

[1]Since sometimes *relations* may refer to relation triplets, to avoid confusion, here we use the term *relation type*.

answer entities whose relation path best matches the question. Different from previous methods, instead of confining the first linking step to only named entities in the question, we consider a wider range of KB units to be linked to the question. We also propose a generation-and-scoring strategy such that we can gradually refine the linked KB units.

Specifically, we divide our method into the following three steps: (1) **Topic unit generation**. In this step, our goal is to identify all KB units that are likely mentioned in the question. We want to achieve high coverage without incurring any heavy computation. So in this step we rely mostly on existing entity linking tools as well as string matching, pre-computed corpus statistics and pre-constructed inverted index. The output of this step is an initial set of topic units for a given question. (2) **Topic unit scoring**. In this second step, we want to refine the topic units obtained in the first step by selecting the top ones based on a sophisticated ranking function learned from the training data. The output of this step is a much smaller subset of the initial topic units. (3) **Relation path ranking**. Given the topic units selected in the previous step, we can derive a set of relation paths where each path starts from a topic unit and contains one or multiple hops of relations. We then use a neural network-based scoring function to rank these relation paths. Finally, we pick the entities connected to the top-ranked relation path as the answers to the question.

Our main contributions lie in the first two steps. We regard the third step as a standard procedure to complete the entire KBQA system. The three steps are illustrated in Figure 4.2. In the rest of this section we present the details of each step and describe how we use reinforcement learning to train the entire method in an end-to-end manner.

4.2.3 Topic Unit Generation

The goal of topic unit generation is to identify all KB units that are possibly mentioned in the question. For named entities appearing in the question, we rely on existing entity linking tools to recognize and link them. Here we mainly describe how we identify other KB units possibly mentioned in the question.

A straightforward solution would be to identify those KB units whose textual descriptions contain one of the words in the question. However, exact word matching is too restrictive. Here we use two strategies to relax the matching conditions. One is to allow character-level n-gram matching. Another is to expand the question with additional words that are highly related to some original question words.

Specifically, we do the following to identify topic units that are not named entities:

1. We build an inverted index to map each unique character n-gram (where $n>4$) and each unique word found in the descriptions of all KB units to the corresponding KB units. This allows us to quickly link a character n-gram or a word found in a question to the KB units contain it.

2. Next, we link the question to some highly correlated words based on statistics obtained from the training data. E.g., the word "flow" appearing in the question in Figure 4.2 could be linked to the word "river" in a relation path, which would help us link "flow" from a question to relation types such as "river mouth" in the KB. To achieve this, for those training questions whose topic entities can be identified by existing entity linking tools, we find the relation paths between the topic entities and the ground truth answer entities. We thus obtain a set of (question, relation path) pairs. We then compute the pointwise mutual information (PMI) between each pair of a question word and a relation path word. Note that although the relation paths used here are not always correct, most of them are still relevant to the question, and therefore the mutual information computed can still indicate strongly correlated words.

3. Given a question q, we first use an entity linking tool to identify named entities in q. Then we remove the linked named entities and stop words in q. For the remaining question words, we use the pre-computed PMI values to find

other words highly correlated with one of the question words (using a PMI threshold of 1) and add these additional words to the question.

4. Finally, we find those KB units linked to all the character n-grams and the words inside the expanded question, based on the inverted index built earlier. This is our initial set of topic units for question q, which we denote with $\bar{\mathcal{U}}_q$.

4.2.4 Topic Unit Scoring

The topic unit generating step aims to increase coverage but usually returns a large set of topic units. In the topic unit scoring step, we use a scoring function to derive a distribution over the topic units identified from the previous step with respect to the question.

The probability function is based on a standard linear feed-forward neural network as follows:

$$s(u,q) = \mathbf{w}_1^\intercal \mathbf{f}_u + b_1, \tag{4.1}$$

$$p(u|q) = \frac{\exp(s(u,q))}{\sum_{u' \in \bar{\mathcal{U}}_q} \exp(s(u',q))}, \tag{4.2}$$

where $u \in \bar{\mathcal{U}}_q$, $\mathbf{f}_u$ is a feature vector associated with u, and $\mathbf{w}_1$ and b_1 are parameters to be learned.

The feature vector $\mathbf{f}_u$ is the concatenation of four vectors:

$$\mathbf{f}_u = \mathbf{f}_u^{\text{semantic}} \oplus \mathbf{f}_u^{\text{character}} \oplus \mathbf{f}_u^{\text{category}} \oplus \mathbf{f}_u^{\text{link}}. \tag{4.3}$$

- $\mathbf{f}_u^{\text{semantic}}$ is the output of a neural network-based sequence matching model [64] that measures the semantic relatedness between the topic unit u and the question q. Here the topic unit u is a represented by the embedding vectors of the words in the textual description of u, denoted as $(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_{|u|})$, and q is also represented by the embedding vectors of its sequence of words,

$(\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_{|q|})$. The sequence matching model first uses the attention mechanism to obtain attention weights a_{ij} as follows:

$$\alpha_{ij} = \mathbf{w}_2^{\mathsf{T}}(\mathbf{q}_i \oplus \mathbf{u}_j \oplus (\mathbf{q}_i \odot \mathbf{u}_j)), \tag{4.4}$$

$$a_{ij} = \frac{\exp(\alpha_{ij})}{\sum_{k=1}^{|q|} \exp(\alpha_{kj})}. \tag{4.5}$$

Then for each word $\mathbf{u}_j$ in the topic unit u, we obtain an attention-weighted version of the question as follows:

$$\tilde{\mathbf{q}}_j = \sum_{i=1}^{|q|} a_{ij} \mathbf{q}_i. \tag{4.6}$$

Finally, we match each unit word $\mathbf{u}_j$ with its corresponding question representation $\tilde{\mathbf{q}}_j$, and aggregate the matching results to obtain the vector $\mathbf{f}_u^{\text{semantic}}$:

$$\mathbf{f}_u^{\text{semantic}} = \mathbf{w}_3^{\mathsf{T}} \sum_{j=1}^{|u|} (\tilde{\mathbf{q}}_j \oplus \mathbf{u}_j). \tag{4.7}$$

Essentially $\mathbf{f}_u^{\text{semantic}}$ is a 1-dimensional vector that encodes the knowledge about how well unit u matches question q using the attention-based sequence matching model. Here $\mathbf{w}_2$ and $\mathbf{w}_3$ are parameters to be learned.

- $\mathbf{f}_u^{\text{character}}$ is a 1-dimensional vector that measures the percentage of characters in u that are also found in q.

- $\mathbf{f}_u^{\text{category}}$ is a 3-dimensional one-hot vector indicating the category of the topic unit u, i.e., whether it is a named entity recognized by the entity linking tool, an entity that is a common noun, or a relation type.

- $\mathbf{f}_u^{\text{link}}$ is a 1-dimensional vector, which contains the entity linking score returned by the entity linking tool if u is a named entity and 0 otherwise.

Based on Eqn. (4.2), we can rank the topic units in $\bar{\mathcal{U}}_q$ and pick the top-K to form a new set $\mathcal{U}_q$ for the next step.

4.2.5 Relation Path Ranking

Given the set $\mathcal{U}_q$, in the relation path ranking step we first identify candidate relation paths that are connected to some $u \in \mathcal{U}_q$ and then rank these relation paths based on how well they match the question. Note that this step is not the focus of our work and we omit some of the implementation details.

To identify the candidate relation paths, for each $u \in \mathcal{U}_q$ we extract a set of relation paths. If u is an entity, we take those relation paths starting from u and containing one or two relations. If u itself is a relation type, we take all relation paths in the KB with one or two relations where at least one of the relations is u. When a unit u gives us more than 500 relation paths, we use character-level overlap with the question q to rank these relation paths and take only the top 500. In the end, we take the union of all the extracted relation paths from all $u \in \mathcal{U}_q$. Let us use $\mathcal{C}_q$ to denote this set of candidate relation paths.

Next, we would like to obtain a distribution over the paths inside $\mathcal{C}_q$. We again use a standard linear feed-forward neural network for this:

$$\begin{aligned} s(c,q) &= \mathbf{w}_4^{\intercal}\mathbf{f}_c + b_4, \\ p(c|q) &= \frac{\exp(s(c,q))}{\sum_{c' \in \mathcal{C}_q} \exp(s(c',q))}, \end{aligned} \tag{4.8}$$

where $c \in \mathcal{C}_q$, $\mathbf{w}_4$ and b_4 are parameters to be learned, and

$$\mathbf{f}_c = \mathbf{f}_c^{\text{link}} \oplus \mathbf{f}_c^{\text{semantic}} \oplus \mathbf{f}_c^{\text{pattern}} \oplus \mathbf{f}_c^{\text{answer}}. \tag{4.9}$$

The four feature vectors are defined as follows:

- For a candidate path c, we consider all its components that are topic units inside $\mathcal{U}_q$ and define $\mathbf{f}_c^{\text{link}}$ to be a 1-dimensional vector containing the sum of the probabilities of these units as computed by Eqn. (4.2).

- $\mathbf{f}_c^{\text{semantic}}$ is based on a previous work on KBQA [83]. It is the output of a sequence matching model that matches the question sequence with the relation path sequence, where the relation path sequence contains the words from the descriptions of the components of the relation paths and a CNN network with a max-pooling layer is used to encode each sequence into a vector before the dot product of the two vectors is computed to measure their similarity.

- $\mathbf{f}_c^{\text{pattern}}$ is also based on some existing work on KBQA [22]. It also uses the same sequence matching model [83] to measure the similarity between the question and a relation path, but entities in the question and the relation path are replaced by placeholders. E.g., “where did Morgan Freeman graduate” becomes (where, did, ⟨e⟩, graduate).

- Since previous work [78] has shown that contexts in the KB of candidate answer entities are useful for KBQA, here we adopt this idea to define $\mathbf{f}_c^{\text{answer}}$. For the relation path c, we collect those entities linked to the answer entities that characterize them. E.g., from the answer entity “Jamie Dornan” we can collect entities “person” and “actor,” which are linked to “Jamie Dornan” in the KB. We call these answer contexts. We then use another CNN network to match the question pattern with these answer contexts to derive $\mathbf{f}_c^{\text{answer}}$.

4.2.6 End-to-End Learning

The model parameters we need to learn include those used in the scoring function at the topic unit scoring step (which we denote as θ_1) and those in the scoring function at the relation path ranking step (which we denote as θ_2). Although these are two separate steps, we jointly learn the parameters in an end-to-end manner. (See Algorithm 1.)

We define the overall loss function as follows. For each relation path, we treat the set of the connected tail entities as the predicted answer. Given a training question q and its ground truth answers, we can compute the F1 score of each relation path $c \in \mathcal{C}_q$. We normalize these F1 scores over all paths and treat it as an empirical

Algorithm 1 Model training

```
1: Input: KB, training questions $\mathcal{Q}$ and their answers
2: Output: $(\theta_1,\theta_2)$
3: Initialize: $(\theta_1,\theta_2)\leftarrow$ pre-trained models (see Section 4.2.7)
4: for each $q\in\mathcal{Q}$ do
5:     Identify the initial set $\bar{\mathcal{U}}_q$ according to Section 4.2.3
6:     Sample $K$ topic units $\hat{\mathcal{U}}_q$ according to $p_{\theta_1}(u|q)$
7:     Use the topic units $\hat{\mathcal{U}}_q$ to identify relation paths $\mathcal{C}_q$
8:     Rank the relation paths using $p_{\theta_2}(c|q)$ and pick the top one to extract the
       answers
9:     Compute the reward $r(\hat{\mathcal{U}}_q)$ based on the F1 score of the answers
10:    Update $\theta_1$ through the policy gradient according to Eqn. (4.11)
11:    Update $\theta_2$ through the gradient of the loss according to Eqn. (4.10)
```

distribution over $\mathcal{C}_q$, which we denote as $\hat{p}(c|q)$. We use the KL-divergence between $\hat{p}(c|q)$ and the predicted distribution $p(c|q)$ from Eqn. (4.8) to measure the loss on q, and we sum over all the training questions in our training set $\mathcal{Q}$ as the total loss:

$$L(\theta_1,\theta_2,\mathcal{Q})=-\sum_{q\in\mathcal{Q}}\sum_{c\in\mathcal{C}_q}\hat{p}(c|q)\log\frac{p(c|q)}{\hat{p}(c|q)}. \tag{4.10}$$

Because at the topic unit scoring step we pick the top-K topic units, which is a discrete choice, the loss function above is not differentiable over θ_1. We thus adopt reinforcement learning and use policy gradient to learn θ_1 [77]. Specifically, let $r(\hat{\mathcal{U}}_q)$ denote the reward of selecting a set of K topic units $\hat{\mathcal{U}}_q\subset\bar{\mathcal{U}}_q$ as the final topic units, the gradient for θ_1 is

$$\nabla_{\theta_1}J(\theta_1)=\mathbb{E}[r(\bar{\mathcal{U}}_q)\cdot\nabla_{\theta_1}\log p_{\theta_1}(u|q)]. \tag{4.11}$$

For the reward $r(\bar{\mathcal{U}}_q)$, we use the final F1 score of the extracted answers when $\bar{\mathcal{U}}_q$ is selected. We use the sampling method to estimate the expected reward. To update θ_2, we fix θ_1 and use the loss function shown in Eqn. (4.10).

Method	**WQSP**	**CWQ**	**SQ**
Our FullModel	**68.2** / 67.9	**39.3 / 36.5**	**80.3**
SOTA	-/**69.0**	34.2/-	78.1
HR-BiLSTM	- / -	- / -	77.0
GRAFT-Net	67.8 / 62.8	- / -	-
HR-BiLSTM†	62.9 / 62.3	33.3 / 31.2	77.6
GRAFT-Net†	67.8 / 62.5	30.1 / 26.0	-

(a)

		WQSP	**CWQ**	**SQ**
avg size of $\bar{\mathcal{U}}_q$	FullModel	24.5	22.6	194.0
	NEOnly	2.9	9.8	163.6
topic unit recall	FullModel	97.3%	83.1%	97.8%
	NEOnly	93.3%	78.4%	96.7%
answer recall	FullModel	93.4%	52.0%	97.7%
	NEOnly	89.5%	48.0%	96.6%

(b)

Feature	**WQSP**
all features	67.9
without $\mathbf{f}^{\text{semantic}}$	61.5 (-6.4)
without $\mathbf{f}^{\text{character}}$	65.9 (-2.0)
without $\mathbf{f}^{\text{category}}$	66.1 (-1.8)
without $\mathbf{f}^{\text{link}}$	64.9 (-3.0)

(c)

Table 4.1: (a) Comparison with existing methods. The top section shows the performance of our full model. The middle section shows previously reported performance. The last section shows the performance of two existing methods reimplemented by us. (b) Coverage of $\bar{\mathcal{U}}_q$. (c) F1 scores on WQSP when different feature configurations are used in Eqn. (4.2).

4.2.7 Implementation Details

We use S-MART [81] as our entity linking tool. We leverage 300-dimensional GloVe [57] word embeddings at the topic unit scoring step and the relation path ranking step. We use Adam optimizer with an initial learning rate of 0.001. All hidden vectors are 200-dimensional. All hyper-parameters are turned on the development data. During training of reinforcement learning, to initialize θ_1, we distantly train the model with surrogate labels of the topic units by checking whether the unit has a relation path leading to the correct answer. To initialize θ_2, we train a baseline using only topic entities returned by our entity linking tool. We set K to be 3.

Method	**WQSP**	**CWQ**	**SQ**
FullModel	**68.2/67.9**	**39.3/36.5**	**80.3**
NEOnly	64.8/64.0	38.4/34.0	78.2
BL	63.6/62.8	36.3/33.8	77.9

Table 4.2: The main experiment results. The metrics used are the commonly-used ones for each dataset. For WQSP and CWQ the metrics are hits@1/F1. For SQ the metrics are accuracy.

4.3 Experiments

4.3.1 Datasets

We evaluate our KBQA method on three benchmark datasets.

WebQuestionsSP (WQSP): This is a dataset that has been widely used for KBQA [85]. It contains 2848 training questions, 250 development questions and 1639 test questions. **ComplexWebQuestions (CWQ)**: This dataset was introduced by Alon Talmor et al. [69] with the intention to create more complex questions from the WebQuestionsSQ dataset. Questions in this dataset often involve relation paths with more than one relations. CWQ contains 27K, 3K and 3K questions for training, development and test, respectively. **SimpleQuestions (SQ)**: This is another popularly used KBQA dataset, introduced by Antonie Bordes et al. [11]. Questions in this dataset can be answered by single-hop relation paths. SQ contains 76K, 11K and 21K for training, development and test, respectively.

For WQSP and CWQ, the knowledge base used is the entire Freebase. For SQ, the knowledge base used is a subset of Freebase that comes with the SQ dataset, which is called "FB2M." To measure the performance, we follow the standard evaluation metric for each dataset. We use hits@1 and F1 scores for WQSP and CWQ, and accuracy for SQ[2].

4.3.2 Main Results

First, we would like to check whether our ideas to consider a wide range of KB units for topic unit linking and to use the generation-and-scoring strategy work. We use ablation experiments to do the comparison. In Table 4.2, **FullModel** refers to our full model that first links a question to a wide range of KB units and then uses the topic unit scoring function to select a small set of topic units for the subsequent relation path ranking. **NEOnly** refers to a degenerate version of the full model where during the topic unit generation step we only consider named entities, i.e., we only use the topic entities returned by the entity linking tool. However, the topic unit scoring step is still retained and trained using reinforcement learning. **BL** refers to a baseline version of our method where only named entities are considered as topic units (same as in **NEOnly**) and there is no scoring and selection of these topic entities.

From Table 4.2 we have the following findings: (1) **FullModel** consistently works better than **NEOnly** on all three datasets, verifying the effectiveness of including a wide range of KB units as topic units in the topic unit generation step. Note that since both **FullModel** and **NEOnly** have the topic unit scoring step, their performance difference is not due to the neural network-based ranking function. (2) **NEOnly** consistently works better than **BL** on all three datasets, showing that even with only named entities as topic units, it is still beneficial to use the neural network-based ranking function to select the top topic units for the subsequent relation path ranking. (3) The improvement of **NEOnly** over **BL** is not as large as the improvement of **FullModel** over **NEOnly**, suggesting that the idea of using a wide range of KB units for topic unit linking is more important.

[2] For hits@1, we used the official evaluation script at `https://www.tau-nlp.org/compwebq`. For F1, we retrieved the ground truth via SPARQL queries and measured by ourselves.

4.3.3 Comparison with Existing Methods

Next, we compare our method with the state-of-the-art (SOTA) performance on each of the three datasets. NSM [42], SPLITQA [69] and BiLSTM-CRF [58] achieve the state of the art on WQSP, CWQ and SQ, respectively. We show their originally reported results in Table 4.1a. Besides, we also consider two recent methods that have been shown to generally work well for KBQA: HR-BiLSTM [88] and GRAFT-Net [68]. We reimplemented these two methods and report both our results and the originally reported results of these two methods.

From Table 4.1a we can see the following: (1) Our full model outperforms the previous state of the art on CWQ and SQ. On WQSP, in terms of hits@1, our model also achieves the state of the art, while in terms of F1, our model still performs competitively although not as good as NSM. (2) Previous state-of-the-art methods NSM, SPLITQA and BiLSTM-CRF were each tested on a single dataset. It is unclear whether they could perform consistently well on different datasets. Our full model is shown to consistently work well on three datasets.

4.3.4 Further Analyses

Coverage of $\bar{\mathcal{U}}_q$. We would like to check if the initial set of topic units $\bar{\mathcal{U}}_q$ as returned by our method indeed has a higher coverage than traditional entity linking. We therefore compare **FullModel** and **NEOnly** in three aspects. We first look at the average size of $\bar{\mathcal{U}}_q$. We then look at *topic unit recall* of $\bar{\mathcal{U}}_q$. This is defined as the percentage of questions for which $\bar{\mathcal{U}}_q$ contains at least one of the KB units found in the ground truth relation paths (which are provided in the datasets but not used for training in our method). We also look at *answer recall* of $\bar{\mathcal{U}}_q$, which is defined as the percentage of questions for which one of the relation paths derived from $\bar{\mathcal{U}}_q$ leads to the correct answer. We show the numbers in Table 4.1b. We can see that indeed FullModel gives a larger size of $\bar{\mathcal{U}}_q$ in general and can increase the topic unit recall and answer recall.

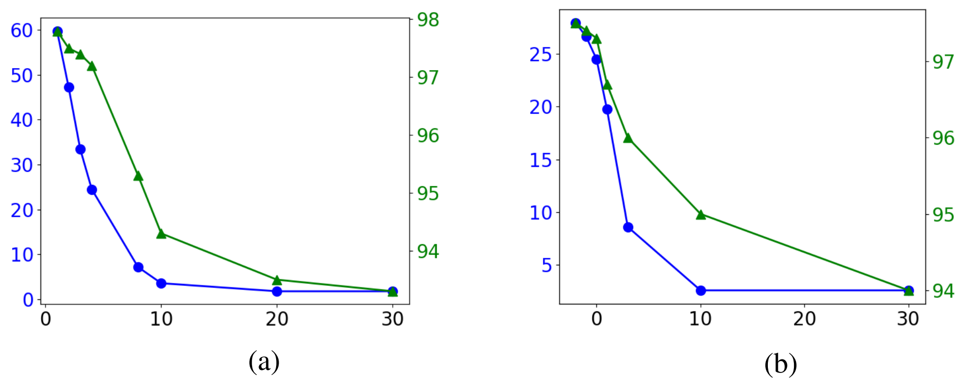


Figure 4.3: Performance on WQSP test in terms of avg size of $\bar{\mathcal{U}_q}$ (in blue) and topic unit recall (in green) when (a) n-gram threshold ranges from 0 to 30 and PMI threshold is 1. (b) n-gram threshold is 4 and PMI threshold ranges from -2 to 30.

Configurations of Topic Unit Generation. In the topic unit generation step, we defined some thresholds, namely, n-gram threshold and PMI threshold. Figure 4.3 shows that with increasing n-gram and PMI thresholds, both average size of $\bar{\mathcal{U}_q}$ and topic unit recall decrease. To obtain scalable topic units without too much decrease of topic unit recall, we set n-gram and PMI thresholds as 4 and 1, respectively.

Features for Topic Unit Scoring. Recall that in the topic unit scoring step our scoring function uses four feature vectors. In Table 4.1c we show the performance in terms of F1 on the WQSP dataset when we use the full model and when we remove each of the feature vectors. We can see that if we remove any of the feature vectors, the performance drops. In particular, the performance decreases the most when the feature $\mathbf{f}^{\text{semantic}}$ is removed, showing the importance of measuring the semantic relevance of a topic unit to the question.

4.4 Conclusions

In this chapter of the dissertation, we propose method that uses topic units for KBQA, which allows us to leverage more information of the questions. We show that our method can achieve either the state of the art or competitive results on benchmark datasets.

Chapter 5

Multi-hop Knowledge Base Question Answering with an Iterative Sequence Matching Model

5.1 Introduction

Previous work on KBQA largely focused on simple questions which can be answered from a single relation connecting two entities in the KB [84, 10, 11, 60]. For example, the question "who is Sylvia Brett's other half" can be answered solely from the triplet (*sylvia brett*,*spouse*,*charles vyner brooke*) in the KB. However, questions in real applications can be more complex and require multiple hops of relations to answer. The question shown in Figure 5.1, for example, requires a relation path of 3 hops in the KB, i.e., (*spouse*→*parent*→*place of birth*), in order to reach a correct answer. Clearly such questions are much harder to handle, because the correct answers are multiple hops away in the KB from the entity appearing in the question, leading to a much larger search space.

We refer to this kind of KBQA problem where the answer entities are multiple hops away in the KB from the entities in the questions the *multi-hop KBQA* problem. Recently, there have been a few attempts to tackle the multi-hop KBQA problem,

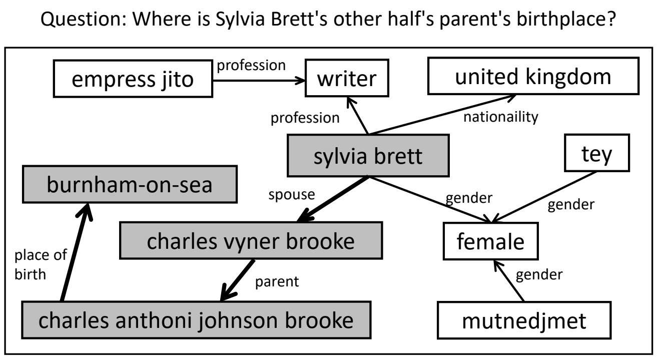


Figure 5.1: An example question and a subset of a KB that contains the answer to the question. The topic entity from the question is "sylvia brett". The shaded boxes show the path of entities and relations that leads to the correct answer.

together with a few benchmark datasets released. For example, Zhang et al. [92] proposed a variational reasoning method that recursively traverses the entities in a KB to predict their probabilities as correct answers. Zhou et al. [94] proposed an interpretable reasoning network for multi-hop KBQA.

Although these studies proposed novel ideas to address multi-hop KBQA and showed promising results, they have a number of limitations. In this chapter of the dissertation, we propose an iterative sequence matching model to address these limitations and empirically show that our method can outperform the previous methods. First, the two existing studies [92, 94] consider a large number of candidate answers or candidate relation paths (albeit in a probabilistic way). This not only makes the methods less efficient but also makes it harder to rank the candidates, because the model has to learn to separate the correct answers from many competing wrong answers. In contrast, we propose to iteratively grow the candidate relation paths that will eventually lead to the candidate answers, and at each iteration we prune away branches that are unlikely to lead to a correct answer. This allows our model to focus on differentiating the correct answers from only those competing candidates that are the most confusing. Our experiments show that indeed this iterative pruning approach works better than a baseline that considers all candidates.

Second, when it comes to matching a question with a relation path that leads to a candidate answer, Zhang et al. [92] and Zhou et al. [94] encoded both the question and the candidate answer as a single embedding vector, and then the two vectors' dot product is computed to measure their similarity. Several previous studies have shown that this kind of sequence matching is not as effective as a match-aggregate sequence matching framework [56, 74, 75]. In our method, we adopt a match-aggregate framework to match the question with a candidate answer's sequence representation. In addition, because we iteratively grow the relation paths leading to candidate answers, we propose a novel incremental sequence matching model to efficiently compute the sequence matching scores without having to revisit the earlier relations in a relation path. Our experiments show that indeed this incremental sequence matching mechanism works better than standard sequence matching.

Third, the two previous studies made some strong assumptions about the problem setup. Zhang et al. [92] assumed that the number of hops needed to answer a question is known in advance, which seriously limits the applicability of the method. In contrast, we propose a mechanism to automatically determine the number of hops needed. Zhou et al. [94] proposed two versions of their method, and for the better performing version, it is assumed that the sequences of relations leading to the correct answers are known during training time. In contrast, our method does not require such information for training and yet we can outperform their method.

Specifically, our method consists of three modules: iterative path growth, incremental sequence matching, and termination check. The three modules work together to iteratively consider relation paths of 1 hop, 2 hops, etc. and iteratively match these paths with the question in order to rank these paths. The method automatically detects when to terminate the iterations. We conduct experiments on three recently released multi-hop KBQA datasets and find that our method performs significantly better than existing methods. We also conduct ablation studies to analyze the contributions of the different components of our method, and we find that both the idea of iteratively pruning the relation paths and the idea of incrementally computing the matching scores are important for our performance gain.

5.2 Our Method

5.2.1 Problem Definition

We assume that a knowledge base (or knowledge graph) is defined over a set of entities $\mathcal{E}$ and a set of relations $\mathcal{R}$. The knowledge base is a set of triplets, which we use $KB = \{(e,r,e')\}$ to represent, where $e,e' \in \mathcal{E}$ and $r \in \mathcal{R}$. We also assume that each entity $e \in \mathcal{E}$ or relation $r \in \mathcal{R}$ has a sequence of words as its textual representation.

A question q is a sequence of words. We assume that entity detection and linking has been done by an entity linking tool and a *topic entity* $e_0 \in \mathcal{E}$ has been identified inside q. Our goal is to find an entity $a \in \mathcal{E}$ that answers the question, and generally speaking, we expect that this answer a is linked to e_0 in the knowledge graph through one or more hops of relations, and these relations between e_0 and a correspond to what is expressed in q. For example, given the question in Figure 5.1, the topic entity is *sylvia brett*. The sequence of relations between *sylvia brett* and the correct answer entity *burnham-on-sea* is (*spouse*,*parent*,*place of birth*), and we can see that these relations collectively correspond to what is expressed in the question.

For training, we assume that we have a set of (q,a) pairs but we do not know which path of relations in the KB leads to the answer a from the topic entity e_0 in q.

5.2.2 Method Overview

The general idea behind our method is to find answer entities that are linked to the topic entity through one or more hops of relations in the knowledge graph. For single-hop KBQA, previous methods typically exhaustively enumerate all the relation paths originating from the topic entity and match them with the question in order to find the best answer. For multi-hop KBQA, exhaustively enumerating all relation paths would lead to too many candidates, which would not only affect efficiency but also making the candidate ranking task harder because of the many competing wrong candidates. Our method iteratively grows the candidate paths and prunes away those paths that are unlikely to lead to the correct answers at each iteration. We also design a novel incremental sequence matching method to score the candidate paths as well as to help check when to terminate the iterations.

Our method consists of three modules: (1) an *iterative path growth* module, (2) an *incremental sequence matching* module and (3) a *termination check* module.

5.2.3 Iterative Path Growth

The iterative path growth module grows the candidate paths up to T hops, one hop at each iteration. At the end of each iteration, it keeps only the top-K candidate paths that best match the question.

Let us first define some terms and notation to facilitate the discussion.

Candidate path: Formally, given a question q, we first detect its topic entity e_0. A *candidate path* is the sequence of entities and relations along a path that starts from e_0 in the knowledge graph. Let us use $p{=}(e_0,r_1,e_1,r_2,e_2,\dots,r_t,e_t)$ to represent a candidate path with t hops of relations. Here, for all $1{\le}l{\le}t$, $(e_{l-1},r_l,e_l){\in}\mathit{KB}$.

Candidate path set: We define the *candidate path set after the t-th iteration* to be the set of the top-K candidate paths we keep at the end of the t-th iteration of our method. We use $\mathcal{P}^{(t)}$ to represent this candidate path set. Note that each $|\mathcal{P}^{(t)}|{=}K$ and each $p{\in}\mathcal{P}^{(t)}$ has t hops of relations.

Tail entity: The tail entity of a candidate path, denoted as *tail*(p), is the last entity in the candidate path p.

Our iterative path growth module works as follows. In the beginning, starting from e_0, we identify all the relations linked to e_0 in the knowledge graph. This gives us the initial candidate path set $\mathcal{P}^{(1)}$. Subsequently, at the t-th iteration, for each $p{\in}\mathcal{P}^{(t-1)}$, we identify all the relations linked to *tail*(p) in the knowledge graph and use them to grow p by one hop of relation. This gives us multiple new candidate paths, each with t hops of relations. Call this set of candidate paths $\tilde{\mathcal{P}}^{(t)}$. We then use the sequence matching module (presented in the next section) to score and rank

Algorithm 2 Iterative Path Growth

```
1:  Input: KB, question $q$, topic entity $e_0$, number of hops $T$
2:  Output: $\mathcal{P}^{(T)}$
3:  Initialize: $\mathcal{P}^{(0)} \leftarrow \{(e_0)\}$
4:  for $t = 1, 2, \ldots, T$ do
5:      $\tilde{\mathcal{P}}^{(t)} \leftarrow \emptyset$
6:      for each $p \in \mathcal{P}^{(t-1)}$ do
7:          $e_{t-1} \leftarrow tail(p)$
8:          for each $(e, r, e') \in KB$ such that $e = e_{t-1}$ do
9:              $p' \leftarrow p \oplus (r, e')$                  ▷ sequence concatenation
10:             $\tilde{\mathcal{P}}^{(t)} \leftarrow \tilde{\mathcal{P}}^{(t)} \cup \{p'\}$
11:     score and rank elements in $\tilde{\mathcal{P}}^{(t)}$
12:     $\mathcal{P}^{(t)} \leftarrow$ top-$K$ elements in $\tilde{\mathcal{P}}^{(t)}$
```

the paths in $\tilde{\mathcal{P}}^{(t)}$, and keep the top-$K$ paths to form $\mathcal{P}^{(t)}$.

For example, given the topic entity *sylvia brett* in Figure 5.1, we first construct $\mathcal{P}^{(1)}$ that contains the following candidate paths: (*sylvia brett*, *profession*, *writer*), (*sylvia brett*, *nationality*, *united kingdom*), (*sylvia brett*, *gender*, *female*), (*sylvia brett*, *spouse*, *charles vyner brooke*). Next, during the second iteration, we grow each of these candidate paths to construction $\tilde{\mathcal{P}}^{(2)}$. Given the subset of the KB shown in Figure 5.1, $\tilde{\mathcal{P}}^{(2)}$ contains the following candidate paths: (*sylvia brett*, *profession*, *writer*, *profession*$^{-1}$, *empress jito*),[1] (*sylvia brett*, *gender*, *female*, *gender*$^{-1}$, *tey*), (*sylvia brett*, *gender*, *female*, *gender*$^{-1}$, *mutnedjmet*), (*sylvia brett*, *spouse*, *charles vyner brooke*, *parent*, *charles anthoni johnson brooke*). But after pruning away the less relevant branches, $\mathcal{P}^{(2)}$ may contain only (*sylvia brett*, *spouse*, *charles vyner brooke*, *parent*, *charles anthoni johnson brooke*).

The algorithm is formally defined in Algorithm 2.

5.2.4 Incremental Sequence Matching

The objective of the incremental sequence matching module is to assign a score to each candidate path p such that we can rank the paths in $\tilde{\mathcal{P}}^{(t)}$. The score should reflect how well a path p matches the question q.

Since both the question and a candidate path are sequences, normally we could

[1]We use a single symbol "$^{-1}$" to denote the reverse of a relation.

employ a standard sequence matching model such as the ones commonly used in natural language inference [56], paraphrase detection [75] and machine comprehension [52]. However, because our candidate paths grow iteratively, at the t-th iteration, when we need to match a candidate path $p \in \tilde{\mathcal{P}}^{(t)}$ with the question, the prefix of p up to the $(t-1)$-th relation has already been matched with the question in previous iterations. Therefore, it makes sense for us to only match the last relation of p with the question and aggregate the matching score at this iteration with the matching scores from previous iterations. Therefore, our sequence matching mechanism is *incremental*.

We first further introduce some notation to facilitate our discussion.

- To enable sequence matching, we represent question q as $\mathbf{Q} = (\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_m)$, where $\mathbf{q}_i$ is the embedding vector of the i-th word in q.

- For a candidate path $p = (e_0, r_1, e_1, \ldots, r_t, e_t)$, we represent it as $\mathbf{P}^{(t)} = (\mathbf{w}_{r_t,1}, \mathbf{w}_{r_t,2}, \ldots, \mathbf{w}_{r_t,n})$, where $\mathbf{w}_{r_t,j}$ is the embedding vector of the j-th word in the textual representation of r_t.[2] Note that here we only consider r_t and ignore the other relations in p because our matching mechanism is incremental, i.e., at the current iteration, we should focus on the matching between r_t and q. Also note that we ignore the entities $e_1, e_2, \ldots$ along the path because we do not expect these entities to be mentioned in the question. Our preliminary experiments also confirmed that including these entities was not useful.

- For $p \in \tilde{\mathcal{P}}^{(t)}$, let $p_{(t-1)}$ denote the prefix of p up to the $(t-1)$-th relation. That is, if $p = (e_0, r_1, e_1, \ldots, r_{t-1}, e_{t-1}, r_t, e_t)$, then $p_{(t-1)}$ is defined as $(e_0, r_1, e_1, \ldots, r_{t-1}, e_{t-1})$.

We now describe the incremental sequence matching module. At the t-th iteration, let p be a candidate path from $\tilde{\mathcal{P}}^{(t)}$ (the set of candidate paths that need to be scored and ranked), and let $\mathbf{P}^{(t)}$ be the representation of p. We first use a standard

[2]For $\mathbf{P}^{(1)}$, we include the words in the textual representations of both e_0 and r_1.

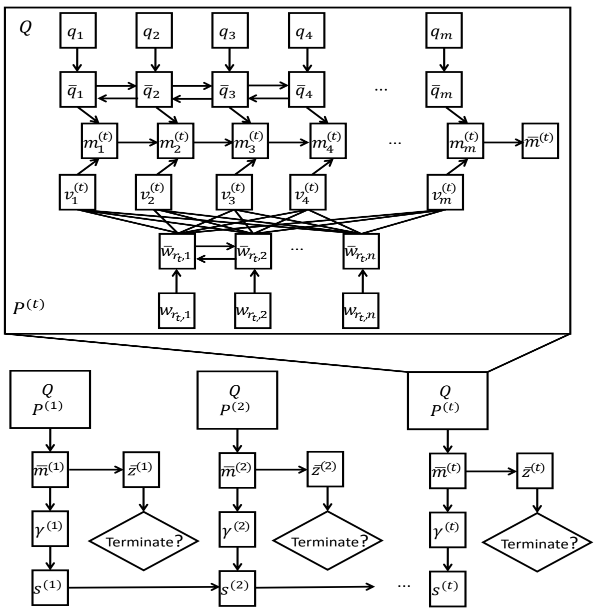


Figure 5.2: The incremental sequence matching model.

BiLSTM to process $\mathbf{Q}$ and obtain $\overline{\mathbf{Q}}=(\overline{\mathbf{q}}_1,\overline{\mathbf{q}}_2,\ldots)$. Essentially $\overline{\mathbf{q}}_i$ represents the i-th word in the question together with its contextual information. We can similarly obtain $\overline{\mathbf{P}}^{(t)}=(\overline{\mathbf{w}}_{r_t,1},\overline{\mathbf{w}}_{r_t,2},\ldots)$ using BiLSTM.

Next, we compute two sets of attention weights, one normalized across $\mathbf{Q}$ and the other normalized across $\mathbf{P}^{(t)}$, as shown below:

$$e_{i,j}=F(\overline{\mathbf{q}}_i)^\intercal F(\overline{\mathbf{w}}_{r_t,j}), \tag{5.1}$$

$$\alpha_{i,j}=\frac{\exp(e_{i,j})}{\sum_{j'=1}^{n}\exp(e_{i,j'})}, \tag{5.2}$$

$$\beta_{i,j}=\frac{\exp(e_{i,j})}{\sum_{i'=1}^{m}\exp(e_{i',j})}, \tag{5.3}$$

where $F(\cdot)$ is a single non-linear layer with ReLU as its activation function.

Now to measure how well $\mathbf{P}^{(t)}$ matches $\mathbf{Q}$, for each word $\mathbf{q}_i$ in $\mathbf{Q}$, we derive an attention-weighted sum of $\overline{\mathbf{P}}^{(t)}$ as follows:

$$\mathbf{v}_i^{(t)} = \sum_{j=1}^{n} \alpha_{i,j} \cdot \overline{\mathbf{w}}_{r_t,j}.$$

Intuitively, we can compare $\overline{\mathbf{q}}_i$ with $\mathbf{v}_i^{(t)}$ to measure how well the i-th word in the question is matched in the current iteration by relation r_t. However, recall that we are doing incremental matching. $\overline{\mathbf{q}}_i$ may have been previously matched with some other words in $p_{(t-1)}$, and if so, it may not be so critical to match $\overline{\mathbf{q}}_i$ with r_t in the current iteration.

To capture this intuition, we borrow an idea from neural machine translation [71, 63], where it is important not to re-translate a word in the source sentence when sequentially generating the target sentence. We define a scalar value $a_i^{(t)}$ to remember how well $\overline{\mathbf{q}}_i$ has been matched in path p up to the t-th iteration. Specifically, we set $a_i^{(0)} = 0$. We then define

$$a_i^{(t)} = a_i^{(t-1)} + \sum_{j=1}^{n} \beta_{i,j}.$$

Note that $\beta_{i,j}$ is as defined above in Eqn. (5.3), based on matching the question with r_t in the t-th iteration. Intuitively, $\sum_{j=1}^{n} \beta_{i,j}$ represents how well $\mathbf{q}_i$ has been matched by all the words in r_t, as compared with other words in $\mathbf{Q}$.

Now with $a_i^{(t)}$ clearly defined, let us define the following matching vector:

$$\mathbf{m}_i^{(t)} = \begin{bmatrix} a_i^{(t-1)} \\ \overline{\mathbf{q}}_i \odot \mathbf{v}_i^{(t)} \\ (\overline{\mathbf{q}}_i - \mathbf{v}_i^{(t)}) \odot (\overline{\mathbf{q}}_i - \mathbf{v}_i^{(t)}) \end{bmatrix}, \qquad (5.4)$$

where $\odot$ represents element-wise multiplication of two vectors. Note that using $\mathbf{v}_1 \odot \mathbf{v}_2$ and $(\mathbf{v}_1 - \mathbf{v}_2) \odot (\mathbf{v}_1 - \mathbf{v}_2)$ to represent how well vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ match has been commonly used in previous work [75, 74]. Here we add $a_i^{(t-1)}$ in the matching vector in order to take previous matching results into consideration. Our preliminary experiments also show that not including $a_i^{(t-1)}$ here would markedly affect the results.

Given the sequence $(\mathbf{m}_1^{(t)},\mathbf{m}_2^{(t)},\ldots,\mathbf{m}_m^{(t)})$, we use an LSTM to process the sequence followed by maxpooling to derive a single vector $\overline{\mathbf{m}}^{(t)}$:

$$\overline{\mathbf{m}}^{(t)}=\text{Max-Pool}(\text{LSTM}(\mathbf{m}_1^{(t)},\mathbf{m}_2^{(t)},\ldots,\mathbf{m}_m^{(t)})). \tag{5.5}$$

We then use this vector to derive a matching score between $\mathbf{Q}$ and $\mathbf{P}^{(t)}$ as follows:

$$\gamma^{(t)}=\mathbf{w}^{\intercal}\overline{\mathbf{m}}^{(t)}+b, \tag{5.6}$$

where the vector $\mathbf{w}$ and scalar b are parameters to be learned.

Note that although $\gamma^{(t)}$ has implicitly encoded the matching results of previous relations in p with the question through $a_i^{(t-1)}$ in Eqn. (5.4), it does not tell us whether each relation in p is critical. When we score the entire path p, we would want to promote those paths in which each segment is highly relevant to the question. To do so, we define the final score for p to be the product of $\gamma^{(t)}$ with $\gamma^{(t-1)}$, $\gamma^{(t-2)}$ and so on. Let $s^{(t)}(p)$ denote the complete matching score between candidate path p and the question. $s^{(t)}(p)$ can be recursively defined as follows: $s^{(0)}(\cdot)=1$. For p with t relations,

$$s^{(t)}(p)=s^{(t-1)}(p_{(t-1)})\cdot\gamma^{(t)}. \tag{5.7}$$

The scoring function $s^{(t)}(\cdot)$ is then used to rank the candidate paths in $\tilde{\mathcal{P}}^{(t)}$ in order to derive $\mathcal{P}^{(t)}$.

Figure 5.2 illustrates how our incremental sequence matching model works.

5.2.5 Termination Check

We now describe how our method determines when to terminate the iterations. Intuitively, if a candidate path p has matched the entire question q well, then we can terminate the iterations. Based on the matching method described above, the vector $\overline{\mathbf{m}}^{(t)}$ encodes how well p matches q. To turn it into a single value to facilitate our termination checking, we first define the following score $z(p) \in [0,1]$ for each candidate path p at the t-th iteration:

$$z^{(t)}(p) = \sigma(\mathbf{v}^{\intercal}\overline{\mathbf{m}}^{(t)} + c),$$

where the vector $\mathbf{v}$ and scalar c are parameters to be learned. Then we take the maximum z among all the paths in $\mathcal{P}^{(t)}$. This implicitly leverages the $z^{(t)}(p)$ of the best-matched candidate path.

$$\overline{z}^{(t)} = \max_{p \in \mathcal{P}^{(t)}} z^{(t)}(p).$$

During training we learn the parameters $\mathbf{v}$ and c for z. During prediction time, we compare $\overline{z}^{(t)}$ with a threshold τ to determine when to stop the iterations.

5.2.6 Loss Function

We now describe the loss function we use during training. The parameters of our model that need to be learned include the following: the parameters of the various LSTMs we use, the parameters of the function $F(\cdot)$ used in Eqn. (5.1), the parameters $\mathbf{w}$ and b in Eqn. (5.6) and $\mathbf{v}$ and c in Eqn. (5.7). We train these parameters in an end-to-end fashion.

For the loss function, we consider two factors. First, we would like the candidate path that leads to the correct answer entity or entities to be ranked higher than the other paths in the candidate path set. Second, we want the value $\overline{z}^{(t)}$ to be close to 1 if we should terminate at the t-hop and close to 0 if we still need to continue to grow the paths.

Specifically, consider the candidate path set $\mathcal{P}^{(t)}$. For each $p \in \mathcal{P}^{(t)}$, by comparing the tail entity (or tail entities if the sequence of relations in p leads to more than one tail entities) with the ground truth answer entity or entities, we can calculate the F1 score of this path p. We then normalize these scores using the following formula:

$$\hat{g}(p) = \frac{\mathrm{F1}(p)}{\sum_{p' \in \mathcal{P}^{(t)}} \mathrm{F1}(p')}.$$

We can think of $\hat{g}(p)$ as the empirical probability for us to choose p among all candidate paths in $\mathcal{P}^{(t)}$.

On the other hand, we derive the probability for p from our model:

$$g(p) = \frac{\exp(s^{(t)}(p))}{\sum_{p' \in \mathcal{P}^{(t)}} \exp(s^{(t)}(p'))},$$

where $s^{(t)}(\cdot)$ is as defined in Eqn. (5.7).

We use the KL-divergence between $\hat{g}(\cdot)$ and $g(\cdot)$ as the first part of our loss function:

$$L_1 = \sum_{p \in \mathcal{P}^{(t)}} g(p) \ln \frac{g(p)}{\hat{g}(p)}.$$

A second goal of our loss function is to help termination check. Based on the ground truth answers, if at iteration t one of the candidate paths in $\mathcal{P}^{(t)}$ can give a F1 score of 1, then we consider t to be the last iteration. We also cap the number of iterations at a constant T. So if t reaches T before we see any F1 score of 1, we also consider this to be the last iteration. Let $\hat{t}$ represent the ground truth number of iterations. We can then define the second part of our loss function as follows:

$$L_2 = -\left(\log(\overline{z}^{(\hat{t})}) + \sum_{t=1}^{\hat{t}-1} \log(1-\overline{z}^{(t)})\right).$$

Our final loss funtion is

$$L = L_1 + L_2.$$

5.3 Experiments

5.3.1 Data Sets

To evaluate the method we have proposed, we conduct experiments using three recently released datasets designed for multi-hop KBQA.

	MetaQA	PathQuestion	WC2014
#Train	100k	5688	6416
#Dev	50k	722	758
#Test	10k	696	780
#Pattern	47	128	12
#Entities	40k	2256	1127
#Relations	18	26	12
#Triplets	134k	4050	3977
% 1-hop	25.3	0	80.5
% 2-hop	38.0	25.4	19.5
% 3-hop	36.7	75.6	0

Table 5.1: Some statistics of the three datasets. The first section shows the number of the questions in different splits and the question patterns. The second section shows the number of the entities, relations and triplets in the associated KB. The third section shows the percentage of the different hop questions.

MetaQA (MoviE Text Audio QA): This dataset was introduced by Zhang et al. [92].[3] It contains more than 400K questions in the movie domain which require either 1-hop, 2-hop or 3-hop reasoning. While the original dataset separates ques-

[3] https://github.com/yuyuz/MetaQA

tions of different hops, we mix all questions together for our evaluation. We take the vanilla text version of the dataset.

PathQuestion: This dataset was used in [94].[4] The questions were created by first identifying the relation paths between pairs of entities in a KB followed by generating natural language questions based on these paths using templates. Some further post-processing was done to vary the questions to make them more real. We mix the 2-hop and 3-hop questions in PathQuestions for our evaluation.

WC2014 (WorldCup2014): This dataset was created by Zhang et al. [91].[5] It contains a mixture of 1-hop and 2-hop questions related to 2014 World Cup.

Some statistics of the datasets are shown in Table 5.1.

In all the original KBs, relation triplets are directional. To simplify our path growth module, we add new edges to the KBs by reversing the direction of each triplet and adding the suffix *INV* to the relation description. For example, from (*sylvia brett*,*profession*,*writer*), we create (*writer*,*profession INV*,*sylvia brett*) and add it the the KB.

5.3.2 Experiment Setup

Although the three datasets already have topic entities annotated, here for fair comparison we perform our own entity linking based on simple string matching. Due to the simple pattern of the question, we have achieved near-perfect entity linking accuracy on the three datasets. We do not provide more details here because entity linking is not the focus of this chapter.

It is worth noting that when we train on the PathQuestion dataset, since there are many 3-hop questions where the answer entity is also directly connected to the topic entity in the KB but the relation between them is not relevant to the question, our method would mistakenly treat these 1-hop connections as correct candidate paths. For example, "The place of birth of parent of Henri Vic-

[4]https://github.com/zmtkeke/IRN
[5]https://github.com/zmtkeke/IRN

tor Regnault's offspring", the golden relation path "*Henri Victor Regnault*$\xrightarrow{\textit{children}}$ *Henri Regnault*$\xrightarrow{\textit{parent}}$*Henri Victor Regnault*$\xrightarrow{\textit{place of birth}}$*Aachen*" and fake golden relation path "*Henri Victor Regnault*$\xrightarrow{\textit{place of birth}}$*Aachen*" both give us correct answers, if we follow the same criteria to supervise our model, the fake relation path cheat the model to obtain an early stop signal. Such scenarios happen a lot in PathQuestion dataset. To avoid this problem, for PathQuestion, we also make use of the ground truth hop numbers to supervise our model. Note that even with this setting on PathQuestion, we are not using any more information for training than previous methods by Zhang et al. [92] and by Zhou et al. [94],

We use the Adagrad optimizer [24] with an initial learning rate of 0.01. We use 300 as the dimension of all word embedding vectors. Word embeddings are initialized via GloVe [57]. All hyper-parameters are tuned on the development data. All hidden dimensions in the model are set to 200 after tuning it among $\{100,150,200,250\}$. The dropout ratio is set to 0 after tuning it among $\{0,0.1,0.2,0.3,0.4\}$. For our iterative sequence matching method, the threshold τ is set to 0.5 after tuning it among $\{0.3,0.5,0.8\}$. Finally, we set $T=3$ and $K=3$.

We use accuracy of the top-1 predicted answer entity as our evaluation metric, where a predicted answer is considered correct if it is one of the ground truth answers. This metric is the same as the one used by Zhou et al. [94] and essentially the same as $\%$ hits@1 used by Zhang et al. [92]. Note that in the case when our top-ranked candidate path in the last iteration leads to more than one answer entities, we randomly pick one of these answer entity as the top-1 predicted answer entity.

5.3.3 Main Results

We first show the comparison of the following methods on the three datasets:

VRN: This is the Variational Reasoning Network method proposed by Zhang et al. [92]. Since their code is not publicly available, we take their reported performance on MetaQA. Note however that their method assumes that the correct number of

	MetaQA	PathQuestion	WC2014
VRN	59.6/-	-/-	-/-
IRN	21.8/9.2	89.8/82.9	92.6/70.5
MemNN	12.0/6.4	87.1/55.6	90.7/46.6
KVMemNN	16.6/6.5	88.0/56.3	90.5/47.1
Ours	**98.6/98.1***	**96.7/96.0***	**99.9/99.9***

Table 5.2: %Hits@1/F1 scores of various methods on the three datasets. Note that for VRN, we took the reported performance in [92], and we do not have its performance on the other datasets. $*$ indicates that the result is statistically significantly better than the best baseline for that dataset at 0.05 significance value based on McNemar test.

hops to answer a question is known, which we do not assume we have except for PathQuestion.

IRN: This is the Interpretable Reasoning Network proposed by Zhou et al. [94]. When re-implementing the IRN method, we realized that this method does not restrict the relation paths to only those that are connected to the topic entities. We therefore implemented an improved version of IRN by imposing such a constraint.

MemNN: This is the Memory Network method proposed by Weston et al. [76] for KBQA. Following work [11], the memory contains all relevant triplets of topic entities. In our implementation, we include triplets up to 3 hops away from the topic entities in the memory.

KVMemNN: This is the Key-Value Memory Network method [50]. It improves MemNN by splitting memory into two parts: key and value. The key stores the subject entity and relation, and the value stores the object entity. When we train this model and MemNN, we set hop number as 3.

Ours: This is our overall method that iteratively searches the space of candidate paths while pruning away branches with low scores. It also uses our incremental sequence matching to score the candidate paths and our termination check mechanism to determine when to stop the iterations.

Table 5.2 shows the comparison between these methods on the three datasets. From the results, we can observe the following: (1) First of all, our method clearly

	1-hop	2-hop	3-hop
VRN	82.0	75.6	38.3
IRN	14.6	10.7	38.2
MemNN	7.0	11.3	16.0
KVMemNN	6.2	12.6	27.9
Ours	96.3	99.1	99.6

Table 5.3: %Hits@1 performance on different questions in MetaQA.

outperforms all the baseline methods on all three datasets consistently [6]. This demonstrates the effectiveness of our method. (2) Our method in general achieves very high accuracy values on all datasets. This is probably because these datasets were to a large extent generated from templates. Using a neural network model with enough complexity and sufficient training data, it is possible for the model to capture the patterns of these templates. (3) We can see that IRN, MemNN and KVMemNN perform poorly on MetaQA although they can perform well on PathQuestion and WC2014. We think there are a few reasons for this. First, the MetaQA dataset has a much larger KB (see Table 5.1) and thus a larger search space. This shows that these methods cannot easily scale with large KB. Second, both the PathQuestion and WC2014 datasets have a large bias to questions with certain hop number (see Table 5.1) while we sample questions of MetaQA evenly. This shows instead of memorizing the major hop number, our method could detect the hop number accurately. Figure 5.3a verifies that our termination check mechanism performs well on these datasets even with a shrunken searching space.

Next, we show the %hits@1 of these different methods on MetaQA when we group questions by the number of hops needed. The results are shown in Table 5.3. We can see that for VRN, the performance is high on 1-hop questions and gradually drops for 2-hop and 3-hop questions. This is reasonable as longer questions are generally harder to answer. For IRN, MemNN and KVMemNN, their performance increases as the number of hops increases. This is because these methods cannot automatically detect the correct number of hops needed, and in our implementation

[6]We acknowledged that the currently published paper [18] could achieve even better results. Since their paper has not been published at that moment, we didn't compare the results with them.

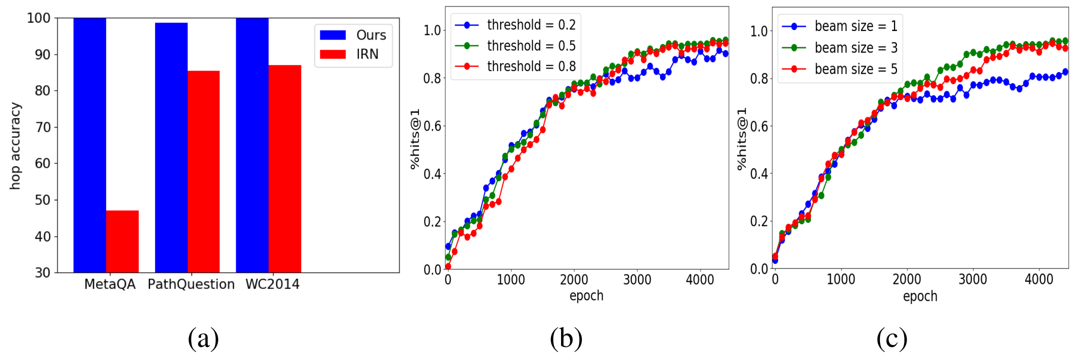


Figure 5.3: (a) Hop number accuracy on three datasets of IRN and Ours methods. (b) Epoch and %hits@1 on test set of MetaQA dataset with thresholds τ=0.2, τ= 0.5, τ=0.8 respectively. (c) Epoch and %hits@1 on test set of MetaQA dataset with beam sizes K=1, K=3, K=5 respectively.

	MetaQA	PathQuestion	WC2014
ES-Siamese	95.7/93.7	89.2/88.8	94.9/90.1
ES-MatchAgg	97.0/96.3	91.5/90.9	96.3/92.1
IS-Prune	97.5/96.5*	92.7/91.8	99.7/99.7*
Ours	**98.6/98.1***	**96.7/96.0***	**99.9/99.9***

Table 5.4: Ablation experiment results. $*$ indicates that the result is statistically significantly better than ES-MatchAgg at 0.05 significance value based on McNemar test.

we force them to always take T=3 hops. Therefore, they end up performing better for 3-hop questions. In contrast, our method automatically detects when to stop the iterations and thus performs consistently well for 1-hop, 2-hop and 3-hop questions.

5.3.4 Ablation Studies

Next, we conduct some ablation studies to test whether each component of our method is necessary. Specifically, we compare with the following variants of our method:

ES-Siamese: This is a basic method that exhaustively searches all candidate paths up to T hops for the best answer entity, i.e., there is no pruning as in our method. When ranking the different paths, a traditional Siamese architecture is used where both the question and the candidate path are each separately encoded into a single vector before the two vectors are matched.

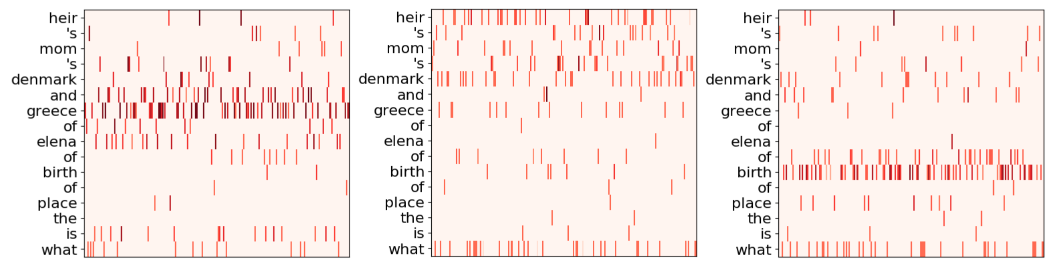
ES-MatchAgg: This method also performs the exhaustive search as ES-Siamese but uses a match-aggregate framework for sequence matching.

Is-Prune: This method uses our iterative path growth mechanism together with the path pruning mechanism. However, it does not use the incremental sequence matching model. Instead, it uses a standard match-aggregate matching method as in ES-MatchAgg. The purpose of this baseline is to measure the effect of pruning.

Table 5.4 shows the ablation experiment results. We can observe the following. (1) ES-MatchAgg is better than ES-Siamese, which shows that match-aggregate sequence matching is useful for our problem. (2) IS-Prune is better than ES-MatchAgg. It is worth noting that pruning is not only to reduce the search space but also to improve the model by restricting the negative candidate paths within the ones that are very similar to the correct candidate paths and the most confusing incorrect answers. Thus our iterative and pruning-based path growth mechanism has the benefit of training a more accurate matching function. (3) Our overall method is better than IS-Prune, showing that the incremental sequence matching mechanism is also effective. It's worth noting that even ES-Siamese baseline could outperform the IRN, MemNN and KVMemNN a lot. We suspect the main reason is that above comparable methods simply represent the question as the unordered bag-of-words and represent the relation or entity by a single embedding without any pre-training. Such methods are not strong enough to select the best-matched relation paths from a large scale of candidates while ES-Siamese baseline can do it using the word-level embedding and expressive encoding method.

5.3.5 Effect of Threshold and Beam Size

Threshold τ and beam size K are two important hyper-parameters in our method. To see how they influence the results, we draw the line plot figures. These figures display the changes of %hits@1 with the increase of training epochs regarding different τ or K. Figure 5.3b displays the effect of the threshold. In Eqn. (5.6), $\gamma^{(t)}$ is a



(a) (b) (c)

Figure 5.4: Visualization of expanded versions of (a) $\overline{\mathbf{m}}^{(1)}$, (b) $\overline{\mathbf{m}}^{(2)}$, (c) $\overline{\mathbf{m}}^{(3)}$ respectively for example question “What is the place of birth of Elena of Greece and Denmark’s mom’s heir?”. The darker color indicates the larger value.

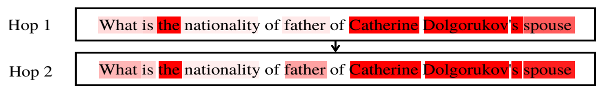

Hop 1
What is the nationality of father of Catherine Dolgorukov's spouse
Hop 2
What is the nationality of father of Catherine Dolgorukov's spouse

Figure 5.5: Visualization of $\mathbf{a}^{(1)}$ and $\mathbf{a}^{(2)}$ for an example sentence after matching with relation *catherine dolgorukov spouse* and *parent* respectively.

indicator which measures how much information of question is already matched by candidate sequences at the t-th iteration, when $\gamma^{(t)}$ is larger than a threshold τ, we terminate the iteration and extract the answer. So when $\tau{=}0.2$, it’s easier to stop the iteration and its performance is relatively good at the beginning, but it falls behind with the increase of training epochs. when $\tau{=}0.8$, it’s harder to stop the iteration at the start of training. Among these three values of threshold, 0.5 achieves best %hits@1 score at last.

In Figure 5.3c, we show the effect of the beam size K. As we can see, when $K{=}3$, the performance increases with relatively fast speed and it achieves the best accuracy at last. However, when $K{=}1$, the accuracy improves relatively slowly and it’s more likely to trap into a local optimum. But increasing K means that we need to sacrifice efficiency, and larger K doesn’t always provide better accuracy (See $K{=}5$). So this is a trade off for us to choose a proper K. In our experiment, we select K as 3.

5.3.6 Visualization

We visualize parts of the incremental matching model to understand its working mechanism.

In Eqn. (5.5), the matched sequence is processed by a LSTM. After that, the most important words are maintained and the other words are filtered by the max-pooling function. In Figure 5.4, we first draw the heatmaps of the output of LSTM. For the position whose value is the maximum along a dimension, we keep the value. Otherwise, we assign 0 to it. If we squeeze the heatmap vertically, we will obtain the exact vectors $\overline{\mathbf{m}}^{(1)}$, $\overline{\mathbf{m}}^{(2)}$ and $\overline{\mathbf{m}}^{(3)}$. So we denote them as expanded versions of $\overline{\mathbf{m}}^{(1)}$, $\overline{\mathbf{m}}^{(2)}$ and $\overline{\mathbf{m}}^{(3)}$. Based on such figures, we could tell which sub-question the model is handling at each iteration. Figure 5.4a displays that at the 1st iteration, the maximum values are mostly distributed among the phrase "elena of greece and denmark's mom". At the 2nd iteration, the maximum values are distributed among "'s heir" and the 3rd iteration has maximum values around "place of birth". This indicates that for a multi-hop question, our model tries to split them to sub-questions and solve sub-questions in the right order.

With similar intuition, we draw the Figure 5.5 to show the value of $a_i^{(t)}$ associated with each of its words for two hops of matching. The words in the dark color are the ones that have been matched up to that iteration. We can see that at the 1st iteration, the phrase "catherine dolgorukov's spouse" has been well matched. At the 2nd iteration, the phrase "father of catherine dolorukov's spouse" has been matched. This accumulative value indeed records how much information of the question we have matched or answered, which could provide clue to decide the termination time.

5.3.7 Error Analysis

Even though our termination check mechanism works well in most cases, we noticed that there are a few wrong stops, which may come from redundant relation matching of the question. For example, in the PathQuestion dataset, the "husband" in the question “what religious belief does Rani Mangammal’s husband have” has been matched twice. Other errors mainly come from incorrect relation matching. Around 74% of the errors with such mis-matching happens at the first hop for 1-hop, 2-hop or 3-hop questions, which might be due to the fact that the first hop relation requires more complex dependency analysis of the question.

5.4 Conclusions

In this chapter of the dissertation, we proposed a novel iterative sequence matching model for multi-hop KBQA. Our method iteratively grows candidate relation paths, prunes away unrelated paths and also automatically detects the end of iterations, which could make the candidate relation path ranking procedure more effective and accurate. Our method achieved state-of-the-art performance on three multi-hop KBQA benchmarks. We further do some analysis to demonstrate the working mechanism of our method [7].

[7] We release our code at `https://github.com/lanyunshi/Multi-hopQA`.

Chapter 6

Future Direction

So far, we have explored multiple downstream applications of KBs. The KB is treated as a powerful structured resource. There remain a lot of potential problems of the KBs that the research communities could investigate in the future.

- **Complex question answering over the KBs**

  KBQA has been studied for decades. Recently, answering complex questions over KBs has become an emerging research problem. For example, answering the question "Which baseball team in American League West was founded first" need more complex inference and reasoning. Linear extension of the graph is not enough. We need to propose more complex searching strategies to find the answers to the complex questions. Meanwhile, controlling the exploding candidate size of the complex question is another challenge. In the future, we are going to focus on solving the more complex question answering over the knowledge bases.

- **Extracting commonsense information from the KBs**

  Our works have shown that the entailment information and entity-relation information in the KB could be leveraged in diverse tasks. Recently, some KBs (e.g., ConceptNet [1]) with commonsense knowledge have attracted re-

[1]`https://conceptnet.io/`

searchers' attention. Such resources could be used in some general tasks. For example, developing commonsense aware dialogue system; generating stories based on commonsense knowledge. Therefore, in the future, it should be a promising direction to explore the applications of the commonsense KBs.

- **Exploring Multimodal relational data in the KBs**

  Recently, the Multimodal KBs [59] have been introduced by some researchers. Besides the numerical and textual attributes (such as ages, dates and descriptions), the images are involved in the KBs as a special attribute. Such image information could play a supplementary role in the KBs. At present, the applications of the Multimodal relational data has not been studied widely. For example, the question answering based on the Multimodal KBs could provide more enriched graphical answers when the user wants the tourist information of a city. Or the user could use the image as part of the questions. It could be interesting to investigate the applications of the Multimodal KBs.

Bibliography

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the International Semantic Web and Asian Conference on Asian Semantic Web Conference*, 2007.

[2] E. Balkir, D. Kartsaklis, and M. Sadrzadeh. Sentence entailment in compositional distributional semantics. In *Proceedings of Conference on International Symposium on Artificial Intelligence and Mathematics*, 2015.

[3] J. Bao, N. Duan, M. Zhou, and T. Zhao. Knowledge-based question answering as machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[4] I. Beltagy, C. Chau, G. Boleda, D. Garrette, K. Erk, and R. Mooney. Montague meets Markov: Deep semantics with probabilistic logical form. In *Proceedings of the Joint Conference on Lexical and Computational Semantics*, 2013.

[5] I. Beltagy, S. Roller, P. Cheng, K. Erk, and R. J. Mooney. Representing meaning with a combination of logical and distributional models. *Computational Linguistics*, 42:763–808, 2016.

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

[7] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[8] M. B. R. Bernardi, N.-Q. Do, and C.-c. Shan. Entailment above the word level in distributional semantics. In *Proceedings of Conference of the European Chapter of the ACL*, 2012.

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

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

[11] A. Bordes, N. Usunier, S. Chopra, and J. Weston. Large-scale simple question answering with memory networks. In *arXiv preprint*, 2015.

[12] A. Bordes, J. Weston, and N. Usunier. Open question answering with weak supervised embedding models. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2014.

[13] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2015.

[14] M. Brysbaert, A. B. Warriner, and V. Kuperman. Concreteness ratings for 40 thousand generally known english word lemmas. 46, 2014.

[15] Q. Cai and A. Yates. Large-scale semantic parsing via schema matching and lexicon extension. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2013.

[16] E. Cambria, D. Olsher, and D. Rajagopal. Senticnet 3: A common and common-sense knowledge base for cognition-driven sentiment analysis. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

[17] Z. Chen, W. Lin, Q. Chen, X. Chen, S. Wei, H. Jiang, and X. Zhu. Revisiting word embedding for contrasting meaning. In *Proceedings of Annual Conference of the Association for Computational Linguistics*, 2015.

[18] Z.-Y. Chen, C.-H. Chang, Y.-P. Chen, J. Nayak, and L.-W. Ku. An unrestricted-hop relation extraction framework for knowledge-based question answering. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 345–356, 2019.

[19] C. Corley and R. Mihalcea. Measuring the semantic similarity of texts. In *Proceedings of ACL Workshop on Empirical Modeling of Semantic Equivalence and Entailment*, 2005.

[20] I. Dagan, O. Glickman, and B. Magnini. The PASCAL recognising textual entailment challenge. *Machine Learning Challenges*, 3944:177–190, 2006.

[21] L. Dong, J. Mallinson, S. Reddy, and M. Lapata. Learning to paraphrase for question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 875–886, 2017.

[22] L. Dong, F. Wei, M. Zhou, and K. Xu. Question answering over freebase with multi-column convolutional neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference of the Asian Federation of Natural Language Processing*, 2015.

[23] Z. Dong, Q. Dong, and C. Hao. Hownet and its computation of meaning. In *Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations*, 2010.

[24] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[25] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2014.

[26] M. Geffet and I. Dagan. The distributional inclusion hypotheses and lexical entailment. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*, 2005.

[27] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2011.

[28] E. Grefenstette. Towards a formal distributional semantics: Simulating logical calculi with tensors. In *Proceedings of Joint Conference on Lexical and Computational Semantics*, 2013.

[29] Y. Hao, Y. Zhang, K. Liu, S. He, Z. Liu, H. Wu, and J. Zhao. An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*, 2017.

[30] J. Herrera, A. Penas, and F. Verdejo. Textual entailment recognition based on dependency analysis and WordNet. In *Proceedings of the 1st PASCAL Recognising Textual Entailment*, 2005.

[31] S. Jain. Question answering over knowledge base using factual memory networks. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

[32] T. M. V. Janssen. Montague semantics. The Stanford Encyclopedia of Philosophy. `http://plato.stanford.edu/archives/win2011/entries/montague-semantics/`, 2011.

[33] W. Julie, W. David, and M. Diana. Characterising measures of lexical distributional similarity. In *Proceedings of International Conference on Computational Linguistics*, 2004.

[34] L. Kotlerman, I. Dagan, I. Szpektor, and M. Zhitomirsky-Geffet. Directional distributional similarity for lexcial inference. *Natural Language Engineering*, 16:359–389, 2010.

[35] T. Kwiatkowski, E. Choi, Y. Artzi, and L. Zettlemoyer. Scaling semantic parsers with on-th-fly ontology matching. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2013.

[36] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer. Neural architectures for named entity recognition. In *Proceedings of Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016.

[37] Y. Lan and J. Jiang. Embedding wordnet knowledge for textual entailment. In *Proceedings of The 27th International Conference on Computational Linguistics*, 2018.

[38] Y. Lan, S. Wang, and J. Jiang. Knowledge base question answering with a matching-aggregation model and question-specific contextual relations. *The Journal of EEE/ACM Transactions on Audio, Speech, and Language Processing*, 27:1629 – 1638, Oct. 2019.

[39] Y. Lan, S. Wang, and J. Jiang. Knowledge base question answering with topic units. In *Proceedings of The 28th International Joint Conference on Artificial Intelligence*, 2019.

[40] Y. Lan, S. Wang, and J. Jiang. Multi-hop knowledge base question answering with an iterative sequence matching model. In *Proceedings of The 19th IEEE International Conference on Data Mining*, 2019.

[41] O. Levy, S. Remus, C. Biemann, and I. Dagan. Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2015.

[42] C. Liang, J. Berant, Q. Le, K. D. Forbus, and N. Lao. Neural symbolic machines: learning semantic parsers on freebase with weak supervision. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*, 2017.

[43] H. Liu and P. Singh. Conceptnet &mdash; a practical commonsense reasoning tool-kit. *BT Technology Journal*, 22(4):211–226, Oct. 2004.

[44] B. MacCartney and C. D. Manning. An extended model of natural logic. In *Proceedings of International Conference on Computational Semantics*, 2009.

[45] K. Mai, T.-H. Pham, M. T. Nguyen, T. D. Nguyen, D. Bollegala, R. Sasano, and S. Sekine. An empirical study on fine-grained named entity recognition. In *Proceedings of International Conference on Computational Linguistics*, pages 711–722, 2018.

[46] F. Manaal, D. Jesse, K. Sujay, D. E. H. Chris, and A. Noah. Retrofitting word vectors to semantic lexicons. In *Proceedings of Conference of the North American Chapter of the ACL*, 2015.

[47] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of International Conference on Language Resources and Evaluation*, 2014.

[48] P. Martinez-Gomez, K. Mineshima, Y. Miyao, and D. Bekki. On-demand injection of lexical knowledge for recognising textual entailment. In *Proceedings of Conference of the European Chapter of the ACL*, 2017.

[49] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of Annual Conference on Neural Information Processing Systems*, 2013.

[50] A. H. Miller, A. Fisch, J. Dodge, and A.-H. Karimi. Key-value memory networks for directly reading documents. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2016.

[51] G. A. Miller. WordNet: A lexical database for english. *Communications of the ACM*, 38:39–41, 1995.

[52] S. Minjoon, K. Aniruddaha, F. Ali, and H. Hananneh. Bi-directional attention flow for machine comprehension. In *Proceedings of the International Conference on Learning Representations*, 2016.

[53] T. Munkhdalai and H. Yu. Neural tree indexers for text understanding. In *Proceedings of the European Chapter of the Association for Computational Linguistics*, 2017.

[54] R. Navigli and S. P. Ponzetto. Babelnet: Building a very large multilingual semantic network. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, 2010.

[55] B. Ofoghi and J. Yearwood. From lexical entailment to recognizing textual entailment using linguistic resources. In *Proceedings of the Australasian Language Technology Association Workshop*, 2009.

[56] A. P. Parikh, O. Taskstrom, D. Das, and J. Uszkoreit. A decomposable attention model for natural language inference. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2016.

[57] J. Pennington, R. Socher, and C. D. Manning. GloVe: Global vectors for word representation. In *Proceedings of Conference on Empirical Methods in Natural Language Processing*, 2014.

[58] M. Petrochuk and L. Zettlemoyer. Simple questions nearly solved: a new upperbound and baseline approach. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 554–558, 2018.

[59] P. Pezeshkpour, L. Chen, and S. Singh. Embedding multimodal relational data for knowledge base completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.

[60] C. Ran, W. Shen, J. Wang, and X. Zhu. Domain-specific knowledge base enrichment using wikipedia tables. In *Proceedings of the IEEE International Conference on Data Mining*, pages 349–358, 2015.

[61] T. Rocktaschel, M. Bosnjak, S. Singh, and S. Riedel. Low-dimensional embeddings of logic. In *Proceedings of the ACL 2014 Workshop on Semantic Parsing*, 2014.

[62] T. Rocktaschel, E. Grefenstette, K. M. Hermann, T. Kocisky, and P. Blunsom. Reasoning about entailment with neural attention. In *Proceedings of International Conference on Learning Representations*, 2016.

[63] A. See, P. J. Liu, and C. D. Manning. Get to the point: summarization with pointer-generator networks. In *Proceedings of the Annual Meeting of Association for Computational Linguistics*, 2017.

[64] M. Seo, A. Kembhavi, A. Farhadi, and H. Hajishirzi. Bidirectional attention flow for machine comprehension. In *Proceedings of International Conference on Learning Representations*, 2017.

[65] L. Sha, B. Chang, Z. Sui, and S. Li. Reading and thinking: Re-read LSTM unit for textual entailment recognition. In *Proceedings of the 26th International Conference on Computational Linguistics*, 2016.

[66] W. Shen, J. Wang, and J. Han. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460, 2015.

[67] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *Proceedings of the World Wide Web Conference*, 2007.

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

[69] A. Talmor and J. Berant. The web as a knowledge-base for answering complex questions. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 641–651, 2018.

[70] Y. Tay, L. A. Tuan, and S. C. Hui. A compare-propagate architecture with aligment factorization for natural language inference. In *arXiv:1801.00102*, 2018.

[71] Z. Tu, Z. Lu, Y. Liu, X. Liu, and H. Li. Modeling coverage for neural machine translation. In *Proceedings of the Annual Meeting of Association for Computational Linguistics*, 2016.

[72] C. Unger, L. Bühmann, J. Lehmann, A.-C. N. Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over rdf data. In *Proceedings of the World Wide Web Conference*, 2012.

[73] S. Wang and J. Jiang. Learning natural language inference with LSTM. In *Proceedings of the Annual Conference of the North American Chapter of the Association for Computational Linguistics*, 2016.

[74] S. Wang and J. Jiang. A compare-aggregate model for matching text sequences. In *Proceedings of International Conference on Learning Representations*, 2017.

[75] Z. Wang, W. Hamza, and R. Florian. Bilateral multi-perspective matching for natural language sentences. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 4144–4150, 2017.

[76] J. Weston, S. Chopra, and A. Bordes. Memory networks. In *Proceedings of the International Conference on Learning Representations*, 2015.

[77] R. J. Williams. Simple statistical gradient following algorithms for connectionist reinforcement learning. *Machine Learning*, pages 8:229–256, 1992.

[78] K. Xu, S. Reddy, Y. Feng, S. Huang, and D. Zhao. Question answering on freebase via relation extraction and textual evidence. In *Proceedings of the Annual Conference of the Association for Computational Linguistics*, 2016.

[79] K. Xu, S. Zhang, F. Yansong, and D. Zhao. Answering natural language questions via phrasal semantic parsing. In *Proceedings of the Natural Language Processing and Chinese Computing*, 2014.

[80] M.-C. Yang, N. Duan, M. Zhou, and H.-C. Rim. Joint relational embeddings for knowledge-based question answering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2014.

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

[82] X. Yao and B. V. Durme. Information extraction over structured data: Question answering with freebase. In *Proceedings of the the Annual Meeting of the Association for Computational Linguistics*, 2014.

[83] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: question answering with knowledge base. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference of the Asian Federation of Natural Language Processing*, 2015.

[84] W.-t. Yih, X. He, and C. Meek. Semantic parsing for single-relation question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2014.

[85] W.-t. Yih, M. Richardson, C. Meek, M.-W. Chang, and J. Suh. The value of semantic parse labeling for knowledge base question answering. In *Proceedings of ACL*, pages 201–206, 2016.

[86] P. Yin, N. Duan, B. Kao, J. Bao, and M. Zhou. Answering questions with complex semantic constraints on open knowledge bases. In *Proceedings of the ACM International Conference on Information and Knowledge Management*, 2015.

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

[88] M. Yu, W. Yin, K. S. Hasan, C. d. Santos, B. Xiang, and B. Zhou. Improved neural relation detection for knowledge base question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

[89] Y. Yu, K. S. Hasan, M. Yu, W. Zhang, and Z. Wang. Knowledge base relation detection via multi-view matching. In *arXiv*, 2018.

[90] L. Zettlemoyer and M. Collins. Learning context-dependent mappings from sentences to logic form. In *Proceedings of Joint conference of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, 2009.

[91] L. Zhang, J. Winn, and T. Ryota. Gaussian attention model and its application to knowledge base embedding and question answering. In *arXiv*, 2016.

[92] Y. Zhang, H. Dai, Z. Kozareva, A. J. Smola, and L. Song. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Conference on Artificial Intelligence*, 2017.

[93] Y. Zhang, S. He, K. Liu, and J. Zhao. A join model for question answering over multiple knowledge bases. In *Proceedings of AAAI Conference on Artificial Intelligence*, 2016.

[94] M. Zhou, M. Huang, and X. Zhu. An interpretable reasoning network for multi-relation question answering. In *Proceedings of the International Conference on Computational Linguistics*, 2018.