# Table Cell Search for Question Answering

Huan Sun[†][*], Hao Ma[#], Xiaodong He[#], Wen-tau Yih[#], Yu Su[†], Xifeng Yan[†]
{huansun, ysu, xyan}@cs.ucsb.edu
{haoma, xiaohe, scottyih}@microsoft.com
[†]University of California, Santa Barbara
[#]Microsoft Research

## ABSTRACT

Tables are pervasive on the Web. Informative web tables range across a large variety of topics, which can naturally serve as a significant resource to satisfy user information needs. Driven by such observations, in this paper, we investigate an important yet largely under-addressed problem: Given millions of tables, how to precisely retrieve *table cells* to answer a user question. This work proposes a novel table cell search framework to attack this problem. We first formulate the concept of a relational chain which connects two cells in a table and represents the semantic relation between them. With the help of search engine snippets, our framework generates a set of relational chains pointing to potentially correct answer cells. We further employ deep neural networks to conduct more fine-grained inference on which relational chains best match the input question and finally extract the corresponding answer cells. Based on millions of tables crawled from the Web, we evaluate our framework in the open-domain question answering (QA) setting, using both the well-known WebQuestions dataset and user queries mined from Bing search engine logs. On WebQuestions, our framework is comparable to state-of-the-art QA systems based on knowledge bases (KBs), while on Bing queries, it outperforms other systems with a 56.7% relative gain. Moreover, when combined with results from our framework, KB-based QA performance can obtain a relative improvement of 28.1% to 66.7%, demonstrating that web tables supply rich knowledge that might not exist or is difficult to be identified in existing KBs.

## Keywords

Question Answering; Table Cell Search; Knowledge Bases

## 1. INTRODUCTION

Tables are straightforward and universal to present relational information. Informative tabular data are pervasive on the Web: According to [27], based on a conservative estimation, over 25 million tables in 500 million Web pages are expressing relational information, as opposed to implementing visual layout. Such tables naturally serve as valuable answer sources to satisfy users' information needs. Tables are also ubiquitous in other realms. For example, enterprises often store their important data about customers, products and employees as tables in spreadsheets or relational databases. Effectively and precisely locating information in these tables are critical to the success of business management and analytics.

Unlike unstructured texts, tabular data provide information in a more structured manner with rows and columns of cells. Table 1 shows a list of countries and their capitals, currencies, and languages. It is an easy task for a human user to find the information she needs when looking at the table. But when there are millions of such tables, manual checking becomes infeasible. How can machines automatically and precisely find information in tables for us? In this paper, we investigate this problem in the setting of open-domain question answering: Users express their information need as a natural language question, and we identify *table cells* from millions of tables on the Web to answer the question. For example, given a question "*What languages do people in France speak*", we retrieve the table cell corresponding to the `MainLanguage`[1] column and the "*France*" row in Table 1, from millions of tables crawled from the Web.

| Country | Capital | Currency | Main Language |
|---|---|---|---|
| Algeria | Algiers | Dinar | Arabic |
| Egypt | Cairo | Pound | Arabic |
| France | Paris | Euro | French |
| ... | ... | ... | ... |

**Table 1: An example table on the Web.**

Question answering (QA) aims at detecting direct answers to natural language questions. Traditional corpus-based QA tries to find answers in plain texts [8, 13, 18, 24, 36, 44]. With the blossom of large open-domain knowledge bases (KBs) like Freebase [7], KB-based QA has attracted much attention recently [5, 6, 17]. Such systems parse a question into a formal representation, e.g., logical form or SPARQL query, to be executed on KBs. However, as noticed in many studies [39, 40, 45], despite their large sizes, existing knowledge bases are still far from complete and not being updated

[*]This research was mainly conducted when the first author was an intern at Microsoft Research.

[1]We will use this font to distinguish column names from other texts.

*WWW 2016,* April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
http://dx.doi.org/10.1145/2872427.2883080.

in a timely fashion [15, 29, 45]. Consequently, information required to answer a question may not always exist in KBs.

Web tables contain a huge body of structured information about wide-ranging topics, and naturally serve as a rich knowledge pool to answer open-domain questions. Employing tables for QA is entitled with the following advantages:

1. Web tables have schema. Different from unstructured texts, each table comes with its own schema; therefore it is much easier to interpret the entities and relations contained in a table. For example, in Table 1, entity "*France*" can be easily interpreted as a country by its column name, and the column name pair <`Country`, `MainLanguage`> is likely to indicate main languages spoken in a country. Such schema and structure provide valuable clues for answering questions.

2. Web tables complement knowledge bases. On the one hand, knowledge bases are incomplete, and web tables might contain information not covered by knowledge bases. On the other hand, the semantic structure of a KB is often more complex than that of a table: For example, Freebase stores the simple fact "*Prescott Bush is George W. Bush's grandfather*" in a complex structure, as shown in Figure 1. On the contrary, tables often represent relations in a more straightforward way, with each column for a relation such as `Father`, `Mother`, `Grandfather`, etc. Such straightforwardness makes answering questions like "*Who is X's grandfather*" much easier.

Question answering based on tables has been studied before in different settings. The problem investigated in Pasupat et al. [31] is related to yet significantly different from ours. In their setting, the table that contains answers to the input question is known beforehand, and their task is to find answers in the given table. In our setting, however, we need to explore a huge set of tables to answer a question. Tables can be viewed from the relational database perspective, which relates our work to the study on natural language interfaces to databases (NLIDBs) [3, 25, 26, 34], where users can pose natural language queries instead of complex SQL queries to access databases. An NLIDB translates a natural language query into an SQL query based on the rigid schema of a given relational database. It is therefore hard to be applied on the unconstrained schemas of web tables in our task, where each table has a self-defined schema. Finally, while in [11, 14] the authors directly search relevant tables to satisfy user queries, we move one step further to precisely find table cells that contain correct answers (i.e., answer cells).

In this paper, we propose an end-to-end framework to identify table cells that can answer a natural language question. The core problem is to match the *unstructured* input question with the *structured* information in tables. We propose a unified chain representation for both the input question and table cells. The *question chain* starts from an identified topic entity in the question (e.g., "*France*"), goes through an edge labeled with the question pattern, and points to the to-be-determined answer. The question pattern is just the input question excluding the topic entity, and expresses the relation between the topic entity and the answer. On the other hand, we also represent the semantic relation between any two cells in the same row of a table as a *relational chain*. For example, the semantic relation

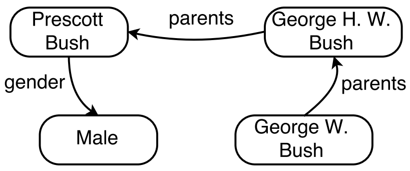


**Figure 1: Representation of the "Grandfather" relation in Freebase is complex.**

between "*France*" and "*French*" in Table 1 is represented as "France$\xrightarrow{\texttt{Country}}\bigcirc\xrightarrow{\texttt{MainLanguage}}$French", where $\bigcirc$ is a pseudo-node referring to the particular row, and will be discussed later in Section 2. Question answering is then reduced to finding the relational chains that can best match with the question chain.

There are two main challenges in identifying the correct answer cells: (1) *Among millions of tables, how to find the relevant ones that may contain answers*? (2) *How to precisely locate answer cells in a relevant table*? With the chain representations, we tackle the first challenge as follows: (a) Detect topic entities in the given question, retrieve tables that contain the topic entities, and generate an initial set of candidate (relational) chains pointing to possible answer cells; (b) Issue the input question to a search engine, and use the returned snippets to select relevant candidate chains from the initial set.

To deal with the second challenge, we develop techniques for more fine-grained matching between candidate chains and the question chain, i.e., to find which candidate chain represents the relation expressed in the question. Simple bag-of-words based matching is insufficient because few words are shared by a candidate chain and the input question. We therefore employ deep neural networks to map both the question and the information about a candidate chain into a common semantic space. We adopt information about a candidate chain from three perspectives: answer type, pseudo-predicate, and entity pairs, which we shall detail in Section 4. We conduct extensive experiments and show that table cells containing correct answers can be effectively identified using the proposed framework. Moreover, combining our table cell search framework with state-of-the-art KB-based QA systems can achieve even better performance, showing that the two different kinds of systems complement each other.

To summarize, our contributions lie in three aspects:

- **Novel Application of Tables**. To the best of our knowledge, our work is among the first attempts to precisely identify *table cells* to answer natural language questions. Being a straightforward way to represent relational information, tables are abundant both on the Web and in enterprise data. How to precisely locate desired information is critical to effectively utilize the rich table resources.

- **Effective Table Cell Search Framework**. We proposed an end-to-end framework to effectively find answer cells for a question. The core concept underlying the proposed framework is the relational chain representation of table cells. With the help of search engine snippets and deep neural networks, we generate and rank candidate chains, and finally extract answer cells from the top ranked chains. Our framework has a close connection to semantic parsing for KB-based question answering, as it can be regarded as parsing a question into a meaning representation using table schemas. Moreover, as shown later, our framework does not rely on any handcrafted grammar, and can be easily extended to closed-domain scenarios such as table cell search in enterprise tables.

- **Extensive Experimental Evaluation**. Based on millions of tables crawled from the Web, we compared our framework with state-of-the-art KB-based QA systems using both the well-known WebQuestions dataset coined on Freebase and a set of free-form questions extracted from Bing query logs. Our framework was evaluated both as a stand-alone system and by being combined with KB-based QA systems. Our experimental results showed that, on WebQuestions, our framework is comparable to the state-of-the-art KB-based QA systems, while on the Bing question set, it outperforms other systems by a relative margin of at least 56.7%. Moreover, when combining our framework with KB-based QA systems, we can achieve even better performance, with a relative improvement of 28.1% and 66.7% respectively. The results verified our hypothesis that web tables supply rich knowledge that does not exist or is too difficult to be utilized in existing KBs.

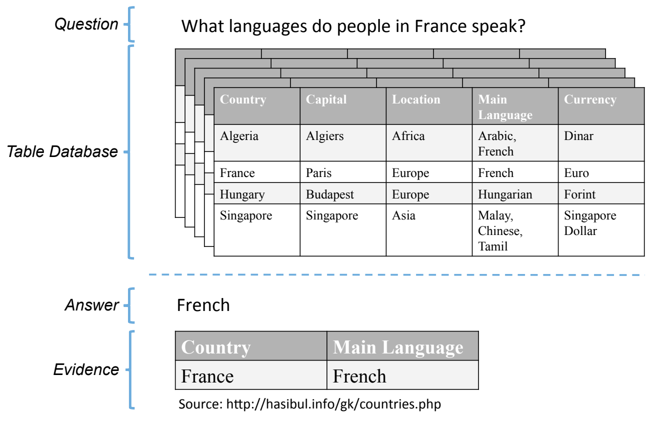
## 2. PRELIMINARIES

In this section, we describe the task addressed in this work, i.e., precisely retrieving table cells among millions of tables to answer natural language questions, followed by the high-level idea of our approach.

### 2.1 Task

Given a natural language question, we aim at answering it by retrieving table cells that contain correct answers, from a large collection of tables. Figure 2 shows a concrete example of this task. For question "*What languages do people in France speak*", one of the tables in the collection denotes several properties of the entity "*France*" in the question. As the column `MainLanguage` can be interpreted as main languages spoken in a country, we would like to identify the table cell "*French*" lying under the `MainLanguage` column and in the same row as "*France*". The identified answer as well as a small sub-table composed of the related cells and column names can be presented, together with the URL of the source table for further exploration.

As implied in this example, we make two assumptions when attacking this task. First, we take an entity-centric view: The targeted questions in this task are those that contain at least one entity, named *topic entity*. This view has been commonly adopted in recent question answering studies [5, 6, 52]. Second, we assume that the relationship between the topic entity in the question and the answer can be represented by the information in a single table. A cell matched with a topic entity and that matched with an answer should occur in the same row. These two assumptions are made because of the unique setting of our task. Using a large collection of web tables as the sole information source for answering questions poses both advantages and challenges, especially when compared to relying on a well-curated knowledge base. For instance, instead of having a rigid schema that defines all possible relations between entities using a fixed set of predicates, as usually seen in a knowledge base, more diversified types of relations are described by a large number of column names existing in millions of tables. Owing to the web redundancy and broad coverage, it's more likely that a pair of column names can match a given question, in contrast to a single predicate in a knowledge base[2]. On the other hand, it is challenging to find relevant tables that might contain answers, from a large collection of independent tables. Diversified forms of relations presented by the column names also increase the difficulty of determining whether they are equivalent to the natural-language description of a question.


Question
What languages do people in France speak?
Table Database
Answer
French
Evidence
Source: http://hasibul.info/gk/countries.php

| Country | Capital | Location | Main Language | Currency |
|---|---|---|---|---|
| Algeria | Algiers | Africa | Arabic, French | Dinar |
| France | Paris | Europe | French | Euro |
| Hungary | Budapest | Europe | Hungarian | Forint |
| Singapore | Singapore | Asia | Malay, Chinese, Tamil | Singapore Dollar |

| Country | Main Language |
|---|---|
| France | French |

**Figure 2: An example of table cell search for QA.**

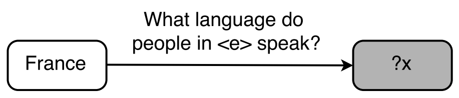


**Figure 3: Question chain: A two-node graph where the topic entity node points to the answer node, and the edge is labeled with the question pattern.**

### 2.2 Approach

Our strategy is to formulate the task as a joint entity and relation matching problem. Here we present the basic idea with intuitive graphical views, leaving more details to be introduced in Section 3.

For each input question, we apply entity linking [50] to identify possible entities in the question. Each identified entity defines the *topic entity* and *question pattern*, where the former is just the canonical name of the entity and the latter is the rest of the question after removing the entity mention. For instance, assuming "*France*" is the topic entity identified in "*What languages do people in France speak*", the question pattern is simply "*What languages do people in <e> speak*", where <e> indicates the slot for the topic entity. A question can then be naturally represented as a two-node graph as in Figure 3. We call this graph a *question chain.* Notice that a question may produce multiple question chains because it can contain more than one topic entity.

Each table essentially defines a "mini knowledge base" – each row describes multi-relations among the cells it con-

[2]More sophisticated analysis of multiple tables and their column names will be needed for complicated, highly compositional questions, which we leave for future work.

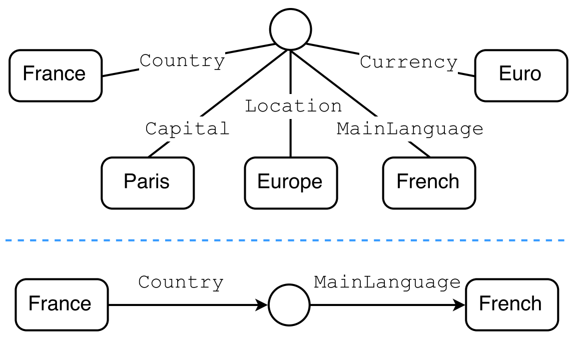


**Figure 4: Top: Each row of a table describes multi-relations among the cells. The circle is a pseudo-node for this particular row, connecting all the cells with their column names as edges. Bottom: A pair of cells form a directional relational chain, which is the path connecting them in the row graph.**

tains. Following [31], a row in a table can be represented by an undirected row graph that connects each cell to a pseudo-node, where the edge is labeled with the corresponding column name (i.e., relation). The pseudo-node[3] simply indicates the row where the cells come from. Figure 4 (top) shows an example of this graph.

A row graph can be decomposed into several *relational chains*. Each relational chain connects two cell nodes by starting from one node, going through the pseudo-node and then pointing to the other node. Similarly, the edges are labeled with the corresponding column names. Figure 4 (bottom) shows one example of this construction. Hereafter, we denote the starting cell of a relational chain as the *topic cell*, and the ending cell as the *answer cell*.

Reminiscent of our second assumption mentioned previously, we shall map the question to a pair of cells in the same row of a table. Having represented the question as a question chain $q$ and a pair of cells as a relational chain $r$, finding a table cell to answer the question is reduced to a chain matching problem. Comparing Figure 3 and 4 (bottom), for $q$ and $r$ to be matched, the topic entity in $q$ has to be matched with the topic cell of $r$, and the question pattern in $q$ needs to be implied by both inward and outward relations of $r$. The answer cell of $r$ can thus be extracted. In case where multiple topic entities are identified, we do not assume all the topic entities simultaneously exist in a single table row; instead, all relational chains starting from a topic cell containing any topic entity will be jointly considered.

## 3. TABLE CELL SEARCH FRAMEWORK

Figure 5 illustrates our end-to-end table cell search framework, which consists of three main steps. We first generate a set of candidate (relational) chains. To do that, we detect topic entities in an input question, match the topic entities with tables to find topic cells, and generate candidate chains from each topic cell. The first step results in a large set of candidate chains, many of which are irrelevant to the question. Therefore, in the second step, search engine snippets providing more information about the input question

[3]The pseudo-node can be analogous to the compound-value-type design in Freebase, which is a standard way to encode multi-relations in RDF triples.

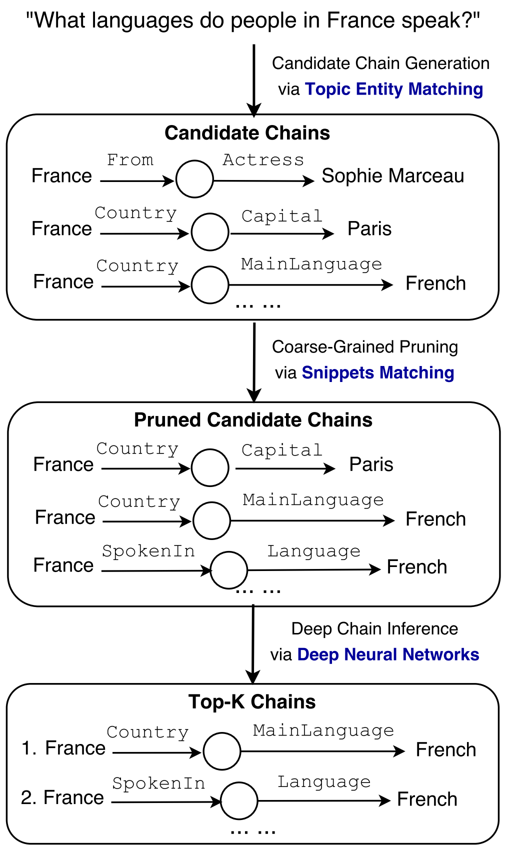

"What languages do people in France speak?"
Candidate Chain Generation
via Topic Entity Matching
Candidate Chains
France From Actress Sophie Marceau
France Country Capital Paris
France Country MainLanguage French
... ...
Coarse-Grained Pruning
via Snippets Matching
Pruned Candidate Chains
France Country Capital Paris
France Country MainLanguage French
France SpokenIn Language French
... ...
Deep Chain Inference
via Deep Neural Networks
Top-K Chains
1. France Country MainLanguage French
2. France SpokenIn Language French
... ...

**Figure 5: Table cell search framework.**

are utilized to help filter out irrelevant chains. We further employ deep neural networks to match the question and a candidate chain in a common semantic space. Overall candidate chains are ranked based on a set of carefully derived features. An answer cell can then be extracted from each top ranked candidate chain.

### 3.1 Candidate Chain Generation

Given a question, one can first apply named entity recognition [30] to identify topic entities, and then retrieve all the table cells that contain any of the topic entities via substring matching. Since an entity often has many aliases (e.g., "*Barack Obama*", "*Barack H. Obama*" or "*President Obama*"), it is therefore beneficial to match table cells with not only the entity mention in the question, but also other entity aliases. Fortunately, open-domain knowledge bases like Freebase store common aliases of an entity; therefore, in our framework, we link each topic entity in the question to Freebase and fetch its alias list. We employ a state-of-the-art entity linking system [50], which is designed particularly for short and noisy texts and has been shown especially suitable for topic entity linking in natural language questions [52]. Table cells containing any alias of a topic entity are retrieved as topic cells. In the rest of the paper, any statement like a table cell *contains* an entity, means the cell contains the entity mention or any of its aliases (if available).

As discussed in Section 2, given a topic cell, we assume the answer cell lies in the same row. But it is unknown which is the answer cell at this stage. We therefore first blindly generate a candidate chain for each possible answer cell, and leave candidate chain ranking for later. Consider Table 1: Assuming we have identified the "*France*" cell under `Country` as a topic cell, three candidate chains are generated, one for each cell in the same row ("*Paris*", "*Euro*", and "*French*"). Each candidate chain starts from the topic cell, goes through the corresponding column names, and points to the candidate answer cell (See Figure 5). We repeat the same procedure for every topic cell, and end up with a large (can be hundreds of thousands) set of candidate chains.

## 3.2 Coarse-Grained Pruning

In the first step, all relational chains related to any identified topic entity are generated as candidates. Consequently, many of them are not truly relevant to the input question, e.g., "France$\xrightarrow{\texttt{From}}\bigcirc\xrightarrow{\texttt{Actress}}$Sophie Marceau" in Figure 5, which is generated from a table about French actresses. We now prune the candidate chain set to obtain a cleaner candidate set for the subsequent ranking model as well as for efficiency consideration. To do that, we need to evaluate the *relevance* of a candidate chain to the input question. However, both the question and a candidate chain usually only contain a few words, and have even fewer words in common. If we directly compare them word-by-word, many relevant chains will be deemed as irrelevant. Therefore, we employ search engine snippets to enrich the question, a common technique used in information retrieval related tasks [40].

We issue the input question $q$ as a query to Bing[4], then compute the word frequency vector based on the top-50 returned snippets, denoted as $w_q$. For each candidate chain $c$, we merge the table caption, topic/answer cells, and column names on it, and then compute its word frequency vector denoted as $w_c$. Two vector similarities are adopted: cosine$(w_c, w_q)$ and InterScore$(w_c, w_q)$ where the latter is defined as $\|w_c \odot w_q\|_0$ and computes the number of unique words in common. Here $\odot$ is the element-wise product and $\|\cdot\|_0$ is the $l_0$ norm of a vector. Candidate chains with both high cosine$(w_c, w_q)$ and InterScore$(w_c, w_q)$ are kept. These two measures inspect vector similarity from different aspects and make a more restrictive selection of relevant candidate chains. If $w_c$ deviates far from $w_q$, the corresponding candidate chain is regarded as irrelevant and thereby discarded.

## 3.3 Deep Chain Inference

After relevant candidate chains to the input question are collected, we perform deeper inference on whether a candidate chain can represent the natural-language statement of the given question. On the candidate chain side, we explore its information from three perspectives: answer type, pseudo-predicate, and entity pairs. We use the question pattern defined in Section 2 to represent what factual information is being asked in the question, regardless of the specific topic entity. In order to capture the syntactically different but semantically equivalent ways of stating the same question, as well as to handle the mismatch between natural language sentences and table schemas, we construct deep neural networks to evaluate the matching degree between a question pattern and each perspective of a candidate chain in a common semantic space. Finally, for each candidate chain, we develop a set of features for downstream ranking so that candidate chains pointing to correct answer cells can be ranked as high as possible. Next we introduce our methodology for deep chain inference in greater details.

[4]`http://www.bing.com/`.

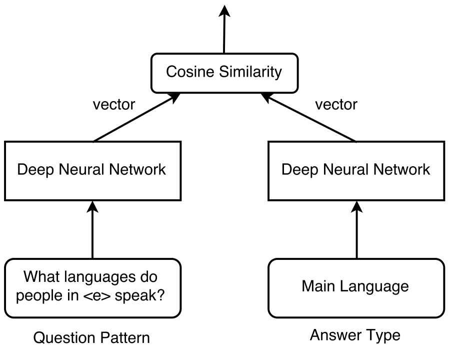


**Figure 6: Semantic similarity between question pattern and answer type in a candidate chain.**

## 4. CHAIN INFERENCE

Coarse-grained pruning gives a set of candidate chains that are likely relevant to the input question. We now need to conduct deeper inference on which candidate chain can actually answer the question. Each candidate chain is inspected from the following perspectives, which give clues about whether the candidate chain matches the question:

(1) Answer type. Answer type is defined as the column name corresponding to the answer cell of a candidate chain. Obviously, the answer type `MainLanguage` matches the question "*What languages do people in France speak*" better than others such as `Currency` and `Capital`.

(2) Pseudo-predicate. While answer type gives information about the answer cell, the relation between the topic cell and the answer cell is also critical for identifying the answer. Predicate is a term representing the relation between two entities in a knowledge base. For example, `PresidentOf` is a predicate between Barack Obama and the United States. Analogically, we use the column name pair, e.g., `Country-MainLanguage`, on each candidate chain to form a *pseudo-predicate*. A pseudo-predicate connects a topic cell to an answer cell and represents a certain relation between them. Intuitively, the pseudo-predicate `Country-MainLanguage` matches questions asking about languages spoken in a country better than other pseudo-predicates such as `Country-Population` and `Country-Currency`.

(3) Entity pairs. Entity pairs from two columns in a table shall have the same relation. For example, all the entity pairs {<Egypt, Arabic>, <France, French>, <Germany, German>, ...} are about some country and its main language. The entity pairs from the same columns as the topic and answer cells in a candidate chain therefore provide significant information about the implicit relation expressed in the chain, complementing the pseudo-predicate.

In cases where any column name is missing on a candidate chain, which is quite uncommon in our experiments, we simply use an empty word set as a replacement. Since the question pattern represents what information is being asked irrespective of the topic entity, intuitively a correct candidate chain should match the question pattern from the above three perspectives. Given the fact that a question pattern usually share few common words with each perspective, we can hardly build effective matching models based on word-level information. For example, the entity pair <Spain, Spanish> shares no common word with the question pattern "*What languages do people in <e> speak*", yet they are about the same relation, i.e., the spoken language of a country. Therefore, we first map them into a common semantic space, where semantically similar texts will be represented as similar fixed-length vectors. Text embedding via neural networks (more broadly termed "deep learning for natural language processing") has been extensively studied recently and demonstrated to excel at capturing the syntactically different ways of stating the same meaning [20, 21, 23, 41, 51, 52]. Hence we employ deep neural networks to embed question patterns and various perspectives about candidate chains and measure their similarity in the semantic space.

Take answer type as example. Figure 6 shows the architecture to match the question pattern with the answer type of a candidate chain. Two deep neural networks are constructed respectively to embed both the question pattern and the answer type. We then compute the cosine similarity of the embedded semantic vectors as the matching degree between the given question and the answer type. The same model architecture is applied to match other perspectives of candidate chains with a question pattern, but model parameters are separately learned for each perspective using the corresponding inputs.

There could be different designs for the deep neural network in Figure 6. We select the Convolutional Deep Structured Semantic Model [5] (C-DSSM) developed in [38] because of its great potential that has been demonstrated in many information retrieval related tasks [20, 37, 52]. Figure 7 illustrates the C-DSSM. It takes a word sequence such as "*What languages do people in <e> speak*" as input. The word hashing layer decomposes a word into a vector of letter-trigrams. For example, word "*speak*" is converted to a bag of letter-trigrams {#-s-p, s-p-e, p-e-a, e-a-k, k-e-#} where "#" is the word boundary symbol. All the unique letter-trigrams in the dataset form the letter-trigram vocabulary of size $N$ and each word will be converted to an $N \times 1$ vector (e.g., $f_t$) with each component being the frequency of a letter-trigram in the word. Following this, a convolutional layer concatenates the letter-trigram frequency vectors in a context window of size 3 and projects it to a local contextual feature vector, e.g., $h_t = \tanh(W_c[f_{t-1}, f_t, f_{t+1}]), \forall t = 1, ..., T$. Then a max pooling layer is deployed to extract the most salient local features and forms a fixed-length global feature vector. This global feature vector is subsequently fed to a non-linear feed-forward neural network layer, which outputs the final semantic representation of the original input word sequence (a question pattern or word sequence representing answer type, pseudo-predicate or entity pairs), i.e., $y = \tanh(W_s v)$.

We instantiate the deep neural network in Figure 6 with C-DSSM. Three matching models shall be learned for question pattern respectively paired with answer type, pseudo-predicate, and entity pairs. To train these semantic matching models, we need to collect three training sets, formed by pairs of question patterns and their true answer type/pseudo-predicate/entity pairs. Unfortunately, no such training sets are readily available. Based on the question-answer pairs in existing QA datasets, our mechanism to construct training sets is as follows: For each question in a question-answer training set, we first match both the topic entities and the answer entities with table cells; the matched table cells are respectively named topic cells and answer cells. Then we extract the relational chains connecting a topic cell to an answer cell in the same row. In order to effectively train the model, we shall obtain a cleaner training set; therefore we conservatively keep only the relational chains with both top-20 $\cos(w_c, w_q)$ and $\text{InterScore}(w_c, w_q)$ scores. We conduct manual checking to decrease mismatch between the relational chains and the question. For each selected chain, we extract its answer type, pseudo-predicate, and entity pairs to respectively pair with the corresponding question pattern, and finally form the training sets.

[5]Publicly available at: `http://research.microsoft.com/en-us/projects/dssm/`.

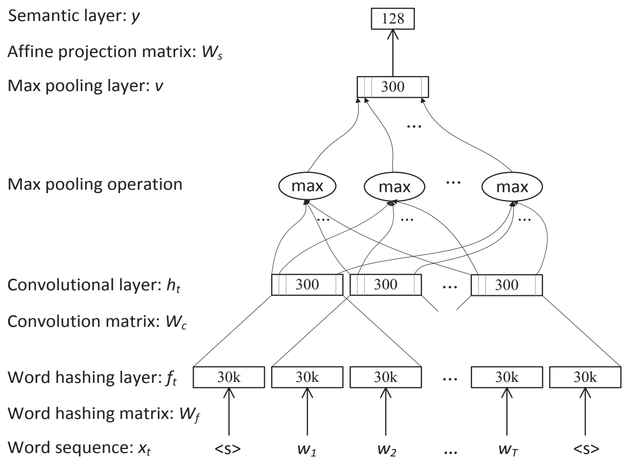


**Figure 7: Architecture of C-DSSM [38]. Number of neurons in each layer is set via a held-out dataset.**

In each case, we randomly sample 5% pairs as the held-out set, and the rest as the training set. Hyper-parameters of the C-DSSM, such as the number of neurons in each layer, are selected using the held-out set. Consistent with other studies employing deep neural networks [22, 52], we observe that the C-DSSM is insensitive to the hyper-parameters in a reasonable range (e.g., 300 $\sim$ 500 nodes in the semantic layer, and learning rate 0.05 $\sim$ 0.005). We leave more details about the C-DSSM related model learning to [22, 37, 38].

## 5. FEATURES

We develop a set of features to rank the candidate chains, which are summarized as below.

### 5.1 Shallow Features

Shallow features consider the matching degree between a question and a candidate chain at the word level.

As introduced in Section 3, for each question, we prepare the word frequency vector $w_q$ based on the top-50 snippets returned by a search engine. On the candidate chain side, we construct the word frequency vector $w_c$ based on its table caption, column names and table cells. Two similarity measures are then applied:

- $\text{cosine}(w_q, w_c) = \frac{w_q \cdot w_c}{\|w_q\|_2 \|w_c\|_2}$
- $\text{InterScore}(w_q, w_c) = \|w_q \odot w_c\|_0$

where InterScore stands for the intersection score and calculates the number of overlapped words in $w_q$ and $w_c$.

## 5.2 Deep Features

Three perspectives are investigated: answer type, pseudo-predicate, and entity pairs. For a candidate chain $c$, the word sequence of the three types of information is denoted as $c_a$, $c_p$, and $c_e$, respectively. With the trained C-DSSM model, we capture the high-level features of $c_a$, $c_p$, and $c_e$ respectively as $y(c_a)$, $y(c_p)$, and $y(c_e)$. On the question side, we input the word sequence representing the question pattern ($q_p$) and extract its high-level features $y(q_p)$. Additionally, we concatenate the topic cell with the pseudo-predicate $c_p$, noted as $c_{p*}$, and compare it with the original question sentence $q$. This feature specifically takes into account the effect of topic entities on semantic matching, which was also adopted in [52].

The semantic similarities between information on the question side and that on the candidate chain side are calculated as below and incorporated as features in our framework:

- DeepType: $\text{cosine}\big(y(q_p), y(c_a)\big)$
- DeepPredicate: $\text{cosine}\big(y(q_p), y(c_p)\big)$
- DeepEntityPairs: $\text{cosine}\big(y(q_p), y(c_e)\big)$
- DeepSentence: $\text{cosine}\big(y(q), y(c_{p*})\big)$.

Here DeepSentence can be regarded as a variation of DeepPredicate. For the word sequence on entity pairs $c_e$, we include two variations: (1) $c_e$ is the word sequence generated by concatenating all entity pairs under the two columns of a candidate chain, in the order of their row indices; (2) $c_e$ is the word sequence corresponding to a single entity pair, and we average $\text{cosine}\big(y(q_p), y(c_e)\big)$ over all entity pairs corresponding to a candidate chain as DeepEntityPairs.

Overall, for each candidate chain, features investigated include shallow features {cosine, InterScore} and deep features {DeepType, DeepPredicate, DeepEntityPairs, DeepSentence }.

## 5.3 Ranking

Based on the above features, we map a candidate chain to a feature vector *w.r.t.* the question. A ranking algorithm shall be deployed to order candidate chains based on their feature vectors. For each question, in the coarse-grained pruning stage, we select candidate chains with both top-3K cosine similarity and top-3K InterScore, in order to reduce noise and speed up the ranking process. For training, we label each candidate chain as correct if its answer cell contains at least one gold-standard answer and incorrect if otherwise. We adopt an in-house fast implementation of the MART gradient boosting decision tree algorithm [9, 19], which learns an ensemble of regression trees and has shown great performance in various tasks [10].

## 6. EXPERIMENTS

Now we evaluate our table cell search framework in the open-domain question answering setting.

## 6.1 Experimental Setup

### *Table Sets*

We test our framework using two sets of tables as answer sources: one is extracted from Wikipedia pages whereas the other is from the broader Web, denoted respectively as WikiTables and AllTables. We employ the table extractor used in [48], which extracts HTML tables from the web crawl and deploys a classifier to distinguish relational tables from other types of tables, such as layout or formatting tables. This approach is also similar to the one used in [12]. We do not discuss the details here since it is not the main focus of this paper. WikiTables contains around 5 million tables whereas AllTables contains roughly 99 million tables, much larger but also noisier than WikiTables.

| WebQ Splits: | WebQ Examples: |
|---|---|
| 2,032 testing | who did the voice for lola bunny? |
| 3,778 training | in what coutries do people speak danish? |

| BingQ Splits: | BingQ Examples: |
|---|---|
| 1,164 testing | cherieff callie voice |
| 4,725 training | boeing charleston sc plant location |

**Table 2: Statistics of question sets.**

| Datasets | WikiTables | AllTables |
|---|---|---|
| WebQ | Training: 2,551 (68%)<br>Testing: 1,362 (67%) | Training: 2,818 (75%)<br>Testing: 1,507 (74%) |
| BingQ | Training: 2,794 (59%)<br>Testing: 679 (58%) | Training: 3,235 (68%)<br>Testing: 793 (68%) |

**Table 3: Table coverage of question sets.**

### *Question Answering Evaluation Sets*

To test our table cell search framework, we pick the popularly studied open-domain QA setting: Open-domain QA datasets and systems are available, which makes the evaluation and comparison with different systems very straightforward. Nevertheless, our framework can be extended to closed-domain question answering as long as the table sources are domain-specific. In our experiments, two question-answer sets are employed, where the gold-standard answer set of each question contains one or more entities in Freebase. We show the statistics and example questions in Table 2.

**WebQ**. WebQuestions (WebQ) is developed by [5] and consists of 5,810 question-answer pairs. The questions are collected using the Google Suggest API and answers are obtained from Freebase with the help of Amazon MTurk. The dataset is split into training and testing sets, respectively containing 3,778 questions (65%) and 2,032 questions (35%). Since its release, WebQ has been widely used in testing an array of open-domain QA systems [5, 6, 16, 51, 52].

**BingQ**. BingQ is constructed in [40]. Questions in this dataset are mined from search engine logs, and therefore not necessarily well-formed yet reflect realistic information needs of the general public. Each question is crowdsourced to obtain correct entity answers. One distinction between BingQ and WebQ is that the knowledge required to answer a question in BingQ (i.e., the relation between its topic entity and answer entity) might not exist in KBs. There are in total 5,889 QA pairs, in which 4,725 (80%) are randomly selected for testing and 1,164 (20%) for training in [40].

**Table-answerable question sets**. The construction of the above evaluation sets does not refer to our table collections, therefore the knowledge required to answer a question might not exist in WikiTables or AllTables. Such questions are unanswerable using the table collections, no matter what

| | WebQ | | | BingQ | | |
|---|---|---|---|---|---|---|
| Features | Precision | Recall | $F_1$ | Precision | Recall[*] | $F_1$ |
| Shallow Features | 0.4214 | 0.3373 | 0.3561 | 0.4035 | 0.4035 | 0.4035 |
| Deep Features | 0.5352 | 0.4210 | 0.4462 | 0.4757 | 0.4747 | 0.4750 |
| Shallow + Deep Features | **0.5712** | **0.4540** | **0.4804** | **0.5817** | **0.5817** | **0.5817** |
| Shallow + DeepType | 0.5433 | 0.4323 | 0.4566 | 0.5493 | 0.5493 | 0.5493 |
| Shallow + DeepPredicate | 0.5492 | 0.4315 | 0.4572 | 0.5493 | 0.5493 | 0.5493 |
| Shallow + DeepSentence | 0.4728 | 0.3768 | 0.3954 | 0.4227 | 0.4227 | 0.4227 |
| Shallow + DeepEntityPairs | 0.4662 | 0.3703 | 0.3907 | 0.5538 | 0.5528 | 0.5531 |
| All Features − DeepType | 0.5551 | 0.4362 | 0.4623 | 0.5538 | 0.5538 | 0.5538 |
| All Features − DeepPredicate | 0.5609 | 0.4474 | 0.4732 | 0.5714 | 0.5714 | 0.5714 |
| All Features − DeepSentence | 0.5639 | 0.4467 | 0.4729 | 0.5803 | 0.5803 | 0.5803 |
| All Features − DeepEntityPairs | 0.5698 | 0.4523 | 0.4786 | 0.5596 | 0.5596 | 0.5596 |

[*] Recall is close to precision because almost every question in BingQ has only one answer.

**Table 4: Performance of different feature combinations.**

algorithms are developed to search table cells. In order to evaluate the effectiveness of our framework, we need to remove those unanswerable questions. It is difficult, if not impossible, to automatically verify whether a question is answerable by a huge set of tables. We adopt the following mechanism to make an approximation: We will keep a question if and only if at least one of its topic entities and at least one of its answer entities simultaneously exist in the same row of some table. Such existence is merely checked by substring match; therefore, it is still a challenging task to evaluate whether the semantic relation between two table cells matches a question or not. Our framework focuses on dealing with this task, and we leave more challenging problems involving joining and integrating tables for future work. The percentage of questions left in each evaluation set is defined as its coverage by a particular table set, which is shown in Table 3. As one can see, a large proportion of questions in both sets, around 58% to 75%, are covered by tables, and AllTables covers roughly 10% more questions than WikiTables. We next evaluate our framework only on questions covered by each table set.

### *Evaluation Measures*

For each question, we extract the answer cell from each of the top-$K$ ranked candidate chains, and thereby top-$K$ answer cells are retrieved. To evaluate the results, we adopt Precision, Recall, and $F_1$ measures as similarly defined in traditional information retrieval [28], since table cell search is analogical to retrieving relevant documents to a query. We regard an answer cell as relevant to the input question if it contains at least one entity in the gold-standard answer set. Therefore, for question $q$, Precision ($P$) and Recall ($R$) are defined as $P = \frac{n_q}{K}$ and $R = \frac{a_q}{N_q}$. Here $n_q$ is the number of *relevant* answer cells retrieved, $a_q$ is the number of unique gold-standard answer entities contained in the top-$K$ answer cells, and $N_q$ is the total number of gold-standard answer entities. $F_1$ is the harmonic mean of $P$ and $R$. We present the average Precision, Recall, and $F_1$ over all test questions as final results.

### *Alternative Systems for Comparison*

KB-based QA systems via semantic parsing can be regarded as a special kind of table cell search if we take a tabular view of knowledge bases: Each predicate corresponds to a two-column table with subject entities in one column and object entities in the other. Two state-of-the-art such systems are taken into account. Sempre and ParaSempre developed in [5] and [6] have shown excellent performance to answer questions in WebQ. We directly use their predicted results on WebQ. On BingQ, we re-train the systems on the training set and evaluate using the testing set. The implementation of the systems is publicly available in `https://github.com/percyliang/sempre`. These systems return a set of entities from Freebase as answer. For fair comparison, the returned entity set is treated as their top-1 retrieved answer cell, and evaluated according to our previously defined measures. Unless otherwise stated, we compare the top-1 answer cell returned by each system in the following experiments, following convention where only the top-1 result is presented to users as an answer.

## 6.2 Performance of Different Feature Groups

We first discuss the performance of different feature combinations in our framework, with WikiTables as the answer source. For each feature combination, we merely use features from that combination for training and testing. As shown in Table 4, we have made the following discoveries:

(1) Shallow features, which only take advantage of word-level information, actually achieve surprisingly good performance on both evaluation sets. This may be explained by the high redundancy of tables and the exploitation of search engine snippets: A table using similar wording as the input question and its snippets likely exists, which makes direct word-level matching effective on some questions. It shows a unique advantage of using web tables as answer sources, as opposed to using a rigidly defined knowledge base. Nevertheless, deep features still get much better performance than shallow features, showing that deeper inference is also necessary and is more advanced.

(2) Shallow features and deep features complement each other. Search engine snippets used in computing shallow features can gather question-related information on the Web to help match a question with the correct candidate chain. Therefore, although shallow features are less effective than deep features, combining them achieves the best performance, with a 34.9% $\sim$ 44.2% relative improvement over only shallow features and 7.7% $\sim$ 22.5% over only deep features on both evaluation sets.

|  | WebQ |  |  | BingQ |  |  |
|---|---|---|---|---|---|---|
| System | Precision | Recall | $F_1$ | Precision | Recall | $F_1$ |
| TabCell | 0.5712 | 0.4540 | 0.4804 | 0.5817 | 0.5817 | 0.5817 |
| Sempre | 0.5419 | 0.4738 | 0.4895 | 0.3328 | 0.3328 | 0.3328 |
| ParaSempre | 0.6013 | 0.5294 | 0.5463 | 0.3711 | 0.3711 | 0.3711 |
| **TabCell +ParaSempre** | **0.7702** | **0.6765** | **0.6998** | **0.6186** | **0.6186** | **0.6186** |

**Table 5: Comparison of different systems.**

(3) Each deep feature defined in Section 4 is combined with shallow features to compare their relative advantage. The two deep features based on answer type and pseudo-predicate, i.e., DeepType and DeepPredicate, are most important in both evaluation sets. Both DeepType and DeepPredicate contain the answer type information, i.e., the column name corresponding to the answer cell. This gives us an important implication that correctly inferring the answer type of a question is critical to finding correct answers. The performance of DeepEntityPairs is quite different on these two evaluation sets. On BingQ, DeepEntityPairs is slightly better than DeepPredicate and DeepType. This is possibly because BingQ questions are not well-formed word sequences (see examples in Table 2), and using a significant number of entity pairs to match them can be more effective than using regular word sequences such as answer type and pseudo-predicate. Although overall DeepSentence and DeepEntityPairs are not as effective as DeepType or DeepPredicate, removing either of them from our framework can hurt the performance, as seen from the last two rows in Table 4.

## 6.3 Comparison with KB-based QA Systems

We now compare our table cell search framework with two state-of-the-art KB-based QA systems Sempre and ParaSempre, which extract answers from Freebase, a large and widely used knowledge base. This comparison can help gain insights about the two answer sources (web tables *vs.* knowledge bases). Apart from separately evaluating each system, we also combine the predicted answer cell from our framework with that from ParaSempre. If our framework and ParaSempre complement each other in answering different questions, the combined results are expected to induce a large performance gain. Results are shown in Table 5, with TabCell referring to our framework. Our observations lie in two aspects: (1) System performance varies on different evaluation sets. On WebQ, ParaSempre obtains the best performance, yet TabCell is still comparable to Sempre. While on BingQ, TabCell outperforms Sempre and ParaSempre respectively by **74.8%** (from 0.3328 to 0.5817) and **56.7%** (from 0.3711 to 0.5817). These significant differences partly result from the evaluation set construction process. Questions in WebQ were coined on Freebase and are guaranteed answerable by Freebase. However, in BingQ, questions were collected from search engine logs and the knowledge required to answer them does not necessarily exist in Freebase. Nevertheless, we can safely draw the conclusion that our framework is at least as effective as state-of-the-art KB-based QA systems according to these evaluation sets. (2) For each question, we combine TabCell and ParaSempre by simply merging the content in each system's top-1 answer cell. Evaluation on the merged answer cell shows around **28.1**% (from 0.5463 to 0.6998) and **66.7**% (from 0.3711 to 0.6186) improvements over ParaSempre on WebQ and BingQ, respectively. The simple combination approach asserts non-decreasing performance; however, such a large performance gain convincingly indicates that table cell search can complement KB-based QA. It verifies our hypothesis that tables contain rich information that might be missing or difficult to be identified in KBs, and our framework presents an effective way to precisely locate such information to satisfy user needs.

## 6.4 Experiments on All Tables from the Web

We not only test our framework with WikiTables as the answer source, but also with AllTables which is around 20 times larger and covers 10% more questions than the former. On the other hand, AllTables is also noisier since general web users compose tables with less attention to table schema or column naming than Wikipedia contributors. Table parsers [48] can be more error-prone when extracting tables from the general webpages. We now compare these two table sets as answer sources using the larger testing set in Table 3, i.e., 1507 questions in WebQ and 793 in BingQ. These testing sets are preferred over the smaller ones, because it allows the opportunity to show advantages of the larger coverage by AllTables. Table 6 shows the results of our framework using all the features. On WebQ, WikiTables can provide better results than AllTables, partly because of the strong connection between Freebase and Wikipedia, i.e., the knowledge stored in Freebase is heavily derived from Wikipedia [1]. Detecting answers to WebQ questions from the smaller and cleaner WikiTables shall be easier than from AllTables. On the other hand, for BingQ questions which are not necessarily answerable by Freebase or Wikipedia pages, AllTables performs better owing to its higher coverage.

| Table Sources | Precision | Recall | $F_1$ |
|---|---|---|---|
| WebQ |  |  |  |
| WikiTables | 0.5162 | 0.4103 | 0.4342 |
| AllTables | 0.4738 | 0.3708 | 0.3923 |
| BingQ |  |  |  |
| WikiTables | 0.4981 | 0.4981 | 0.4981 |
| AllTables | 0.5233 | 0.5226 | 0.5228 |

**Table 6: Comparison of different table sources**

## 6.5 Evaluation on Top-$K$ Answer Cells

Previously we evaluate using only the top-1 answer cell, following the convention in QA. We now evaluate our framework when multiple answer cells are retrieved, as shown in Table 7. WikiTables are used and all features are included. When increasing $K$, Recall shall be non-decreasing while Precision may decrease. The results show that on WebQ, $K = 2$ and $K = 3$ obtain a better $F_1$ score than when only

|  | WebQ |  |  | BingQ |  |  |
|---|---|---|---|---|---|---|
| Top-$K$ | Precision | Recall | $F_1$ | Precision | Recall | $F_1$ |
| $K$=1 | 0.5712 | 0.4540 | 0.4804 | 0.5817 | 0.5817 | **0.5817** |
| $K$=2 | 0.5323 | 0.5306 | **0.4959** | 0.5052 | 0.6412 | 0.5505 |
| $K$=3 | 0.4983 | 0.5819 | 0.4919 | 0.4470 | 0.6766 | 0.5154 |
| $K$=5 | 0.4430 | 0.6268 | 0.4680 | 0.3602 | 0.7090 | 0.4468 |

**Table 7: Evaluation on top-$K$ ranked answer cells.**

considering the top-1 answer cell. Since WebQ questions usually have multiple answers, presenting multiple answer cells could benefit the overall performance with $F_1$ dominantly affected by the increasing Recall. On the other hand, since BingQ questions usually have only one answer and the relevant answer cell for most questions is already ranked in the first place as shown by Precision@$K = 1$, increasing $K$ leads to a lower $F_1$.

## 7. RELATED WORK

We summarize previous related work in three categories: (1) Question answering; (2) Table search, annotation, and integration; (3) Natural language interface to databases.

**Question answering**. Different types of question answering systems based on texts, knowledge bases (KBs), and tables have been investigated. Most earlier QA systems such as [8, 13, 18, 24, 36, 44] mine answers from TREC [44] document collections or the rich web corpus. QuASM proposed in [33] exploits the structure inherent in web documents to boost question answering. QuASM indexes web documents into smaller units (e.g., text tables, HTML tables) to be utilized by a QA system. QuASM did not utilize the schemas of HTML tables to infer answers to a question, instead, they are treated similarly as the entire document when being used for QA. KB-based QA systems parse natural language questions to specific forms such as logic forms, graph queries, and SPARQL queries, which can be executed against KBs to find answers [5, 6, 17, 35, 42, 47, 52, 54]. QA systems developed in [16] investigate both curated KBs such as Freebase and extracted KBs from general corpora, as answer sources to answer a question. Yang et al. [49] find patterns (i.e., aggregations of subtrees) in a knowledge base to compose table answers to keyword queries such as "Washington cities population". Yao et al. [51] propose to associate question patterns with answer patterns described by Freebase with the help of a web-scale corpus. Noticing that KBs are far from complete, Sun et al. [40] develop a framework to detect answers from the web texts, while still utilizing the rich information about entities in Freebase to determine the true answers. Given a table and a question, [31] parses a question according to the schema of the given table and outputs answers accordingly. Our work is most related to [31], but significantly different in that given a question, the table containing answers is not associated; instead, we aim at finding table cells from millions of tables to answer it.

**Table search, annotation, and integration**. Tables have been actively studied in many aspects such as annotation, integration, and search [2, 14, 27, 32, 43, 53]. For example, Limaye et al. [27] annotate table cells with entities, table columns with entity types to which entities in the column belong, and relations that pairs of table columns seek to express, and show the benefits of good annotations to the performance of a web search tool. Similarly, Venetis et al. [43] recover the semantics of tables by annotating a table with a database of class labels and relationships automatically extracted from the Web. Finding tables in a large corpus of heterogeneous tables related to a user table is investigated in [14]. Several types of relatedness are captured including tables that are candidates for joins and tables that are candidates for union. Pimplikar et al. [32] aim at answering table queries which consist of several keyword columns. In [11, 14], the authors directly return relevant tables to satisfy user queries. While retrieving entire tables is desirable in many scenarios such as finding reusable tables and information summarization on a topic, in this paper we focus on precisely locating table cells to answer questions. Table annotation and integration can be regarded as pre-processing steps to facilitate our table cell search framework. We leave in future studies how to involve integration of intermediate results, when using tables to answer more compositional questions. The discovered facts via QA based on tables can be used to complete existing KBs. A similar methodology has been adopted in [46]. In [4], Balakrishnan et al. share lessons and insights in developing a broad set of applications of web tables at Google. While a brief description on using web tables to answer fact seeking queries is provided, few technical details are provided.

**Natural language interface to databases.** Natural language interfaces to databases (NLIDBs), where users can pose natural language queries instead of writing complex SQL queries, have been studied since several decades ago [3, 25, 26, 34]. NLIDBs will translate a natural language question to an SQL query according to the predefined schema of a relational database. It is hard to be directly applied in our task where each table has a self-defined schema. Our work novelly exploits search engine snippets and deep neural networks to locate table cells for QA.

## 8. CONCLUSION

In this paper, we proposed an end-to-end framework to precisely locate table cells in millions of web tables for question answering. Our table cell search framework was compared with state-of-the-art KB-based QA systems. Through extensive experiments, we showed that our framework could outperform other systems by a large margin on real-world questions mined from search engine logs. Our results also supported the hypothesis that web tables are a good complement to knowledge bases, providing rich knowledge missing from existing knowledge bases. In the future, we would like to extend the framework to tackle more compositional questions where integration of multiple tables and columns will be needed, as well as to table cell search in closed-domain scenarios such as enterprise tables.

## 9. REFERENCES

[1] Freebase wiki. `http://wiki.freebase.com/wiki/Wikipedia`.

[2] M. D. Adelfio and H. Samet. Schema extraction for tabular data on the web. *VLDB*, 6(6):421–432, 2013.

[3] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(01):29–81, 1995.

[4] S. Balakrishnan, A. Halevy, B. Harb, H. Lee, J. Madhavan, A. Rostamizadeh, W. Shen, K. Wilder, F. Wu, and C. Yu. Applying webtables in practice.

[5] J. Berant, A. Chou, R. Frostig, and P. Liang. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, pages 1533–1544, 2013.

[6] J. Berant and P. Liang. Semantic parsing via paraphrasing. In *Proceedings of ACL*, 2014.

[7] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *SIGMOD*, pages 1247–1250. ACM, 2008.

[8] E. Brill, S. Dumais, and M. Banko. An analysis of the AskMSR question-answering system. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 257–264. Association for Computational Linguistics, 2002.

[9] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. *Learning*, 11:23–581, 2010.

[10] C. J. Burges, K. M. Svore, P. N. Bennett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. In *Yahoo! Learning to Rank Challenge*, pages 25–35, 2011.

[11] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 1(1):538–549, 2008.

[12] M. J. Cafarella, A. Y. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational web. In *WebDB*. Citeseer, 2008.

[13] J. Chu-Carroll, J. Prager, C. Welty, K. Czuba, and D. Ferrucci. A multi-strategy and multi-source approach to question answering. Technical report, DTIC Document, 2006.

[14] A. Das Sarma, L. Fang, N. Gupta, A. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu. Finding related tables. In *SIGMOD*, pages 817–828. ACM, 2012.

[15] X. L. Dong, K. Murphy, E. Gabrilovich, G. Heitz, W. Horn, N. Lao, T. Strohmann, S. Sun, and W. Zhang. Knowledge vault: A web-scale approach to probabilistic knowledge fusion, 2014.

[16] A. Fader, L. Zettlemoyer, and O. Etzioni. Open question answering over curated and extracted knowledge bases. In *SIGKDD*. ACM, 2014.

[17] A. Fader, L. S. Zettlemoyer, and O. Etzioni. Paraphrase-driven learning for open question answering. In *ACL (1)*, pages 1608–1618, 2013.

[18] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa project. *AI magazine*, 31(3):59–79, 2010.

[19] J. H. Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, pages 1189–1232, 2001.

[20] J. Gao, P. Pantel, M. Gamon, X. He, L. Deng, and Y. Shen. Modeling interestingness with deep neural networks. In *EMNLP*, 2014.

[21] B. Hu, Z. Lu, H. Li, and Q. Chen. Convolutional neural network architectures for matching natural language sentences. In *NIPS*, pages 2042–2050, 2014.

[22] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck. Learning deep structured semantic models for web search using clickthrough data. In *CIKM*, pages 2333–2338. ACM, 2013.

[23] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *CVPR*, 2015.

[24] J. Ko, E. Nyberg, and L. Si. A probabilistic graphical model for joint answer ranking in question answering. In *SIGIR*, pages 343–350. ACM, 2007.

[25] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. *VLDB*, 8(1):73–84, 2014.

[26] Y. Li, H. Yang, and H. Jagadish. Nalix: an interactive natural language interface for querying xml. In *SIGMOD*, pages 900–902. ACM, 2005.

[27] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and searching web tables using entities, types and relationships. *VLDB*, 3(1-2).

[28] C. D. Manning, P. Raghavan, H. Schütze, et al. *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge, 2008.

[29] B. Min, R. Grishman, L. Wan, C. Wang, and D. Gondek. Distant supervision for relation extraction with an incomplete knowledge base. In *HLT-NAACL*, pages 777–782, 2013.

[30] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[31] P. Pasupat and P. Liang. Compositional semantic parsing on semi-structured tables. *ACL*, 2015.

[32] R. Pimplikar and S. Sarawagi. Answering table queries on the web using column keywords. *VLDB*, 5(10):908–919, 2012.

[33] D. Pinto, M. Branstein, R. Coleman, W. B. Croft, M. King, W. Li, and X. Wei. Quasm: a system for question answering using semi-structured data. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 46–55. ACM, 2002.

[34] A.-M. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM, 2003.

[35] S. Reddy, M. Lapata, and M. Steedman. Large-scale semantic parsing without question-answer pairs. *Transactions of the Association for Computational Linguistics*, 2:377–392, 2014.

[36] N. Schlaefer, P. Gieselmann, T. Schaaf, and A. Waibel. A pattern learning approach to question answering within the ephyra framework. In *Text, speech and dialogue*, pages 687–694. Springer, 2006.

[37] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *CIKM*, pages 101–110. ACM, 2014.

[38] Y. Shen, X. He, J. Gao, L. Deng, and G. Mesnil. Learning semantic representations using convolutional neural networks for web search. In *WWW companion*, pages 373–374, 2014.

[39] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *NIPS*, pages 926–934, 2013.

[40] H. Sun, H. Ma, W.-t. Yih, C.-T. Tsai, J. Liu, and M.-W. Chang. Open domain question answering via semantic enrichment. In *WWW*, pages 1045–1055, 2015.

[41] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112, 2014.

[42] C. Unger, L. Bühmann, J. Lehmann, A.-C. Ngonga Ngomo, D. Gerber, and P. Cimiano. Template-based question answering over RDF data. In *WWW*, pages 639–648, 2012.

[43] P. Venetis, A. Halevy, J. Madhavan, M. Paşca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *VLDB*, 4(9):528–538, 2011.

[44] E. M. Voorhees and D. M. Tice. Building a question answering test collection. In *SIGIR*, pages 200–207. ACM, 2000.

[45] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *WWW*, pages 515–526, 2014.

[46] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *WWW*, 2014.

[47] M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, and G. Weikum. Natural language questions for the web of data. In *EMNLP-CoNLL*, pages 379–390. Association for Computational Linguistics, 2012.

[48] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*, pages 97–108. ACM, 2012.

[49] M. Yang, B. Ding, S. Chaudhuri, and K. Chakrabarti. Finding patterns in a knowledge base using keywords to compose table answers. *Proceedings of the VLDB Endowment*, 7(14):1809–1820, 2014.

[50] Y. Yang and M.-W. Chang. S-mart: Novel tree-based structured learning algorithms applied to tweet entity linking. *ACL*, 2015.

[51] X. Yao and B. Van Durme. Information extraction over structured data: Question answering with freebase. In *ACL*, 2014.

[52] W.-t. Yih, M.-W. Chang, X. He, and J. Gao. Semantic parsing via staged query graph generation: Question answering with knowledge base. *ACL*, 2015.

[53] M. Zhang and K. Chakrabarti. Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In *SIGMOD*, pages 145–156. ACM, 2013.

[54] L. Zou, R. Huang, H. Wang, J. X. Yu, W. He, and D. Zhao. Natural language question answering over RDF: a graph data driven approach. In *SIGMOD*, pages 313–324. ACM, 2014.