Lean Question Answering over Freebase from Scratch

Xuchen Yao
kitt.ai *
2157 N Northlake Way
Seattle, WA 98103, USA

Abstract

For the task of question answering (QA) over Freebase on the WEBQUESTIONS dataset (Berant et al., 2013), we found that 85% of all questions (in the training set) can be directly answered via a single binary relation. Thus we turned this task into slot-filling for <*question topic*, *relation*, *answer*> tuples: predicting *relations* to get *answers* given a *question's topic*. We design efficient data structures to identify question topics organically from 46 million Freebase topic names, without employing *any* NLP processing tools. Then we present a lean QA system that runs in real time (in offline batch testing it answered two thousand questions in 51 seconds on a laptop). The system also achieved 7.8% better F_1 score (harmonic mean of average precision and recall) than the previous state of the art.

1 Introduction

Large-scale open-domain question answering from structured Knowledge Base (KB) provides a good balance of precision and recall in everyday QA tasks, executed by search engines and personal assistant applications. The release of WEBQUESTIONS dataset (Berant et al., 2013) has drawn a lot of interest from both academia and industry. One tendency to notice is that the general trend of research is becoming more complex, utilizing various techniques such as semantic parsing and deep neural networks.

We took a radically different approach by heading for the other direction: simplifying the task as much as possible with no compromise on speed and accuracy. We treat the task of QA from Freebase as a two-step problem: identifying the correct topic (*search* problem) and predicting the correct answer (*prediction* problem). The common approach to the first problem is applying basic linguistic processing, such as part-of-speech (POS) tagging and chunking to identify noun phrases, and named entity recognition (NER) for interesting topics. The common approach to the second problem is detailed question analysis, which usually involves parsing. In any case, various components from the natural language processing (NLP) pipeline are usually applied.

With an emphasis on real-time prediction (usually making a prediction within 100 milliseconds after seeing the question), we chose not to use *any* NLP preprocessing – not even POS tagging. Instead we design efficient data structures to help identify named entities to tackle the search problem.

For the prediction problem, we found that given a question and its topic, simply predicting the KB relation between the topic and the answer is sufficient. In other words, we turned QA from Freebase into a slot-filling problem in the form of <*topic*, *relation*, *answer*> tuples: given a *question*, the task is to find the *answer*, while the search problem is to find the *topic* and the prediction problem is to find the *relation*. For instance, given the question what's sweden's currency?, the task can be turned into a tuple of <Sweden, */location/country/currency_used*, Swedish krona>. In Section 3 we address how to identify the topic (Sweden) and in Section 4 how to predict the relation (*/location/country/currency_used*). There are obvious limitations in this task format, which are discussed in Section 6.

Going beyond reporting evaluation scores, we describe in details our design principle and also report performance in speed. This paper makes the follow-

* Incubated by the Allen Institute for Artificial Intelligence.

Proceedings of NAACL-HLT 2015, pages 66–70,
Denver, Colorado, May 31 – June 5, 2015.

ing technical contributions to QA from KB:

- We design and compare several data structures to help identify question topics using the KB resource itself. The key to success is to search through 46 million Freebase topics efficiently while still being robust against noise (such as typographical or speech recognition errors).

- Our algorithm is high-performance, real-time, and simple enough to replicate. It achieved state-of-the-art result on the WEBQUESTIONS dataset. Training time in total is less than 5 minutes and testing on 2032 questions takes less than 1 minute. There are no external NLP library dependencies: the only preprocessing is lowercasing.

2 Related Work

The task of question answering from Freebase was first proposed by Berant et al. (2013), who crawled Google Suggest and annotated 5810 questions that had answers from Freebase with Amazon Mechanical Turk, thus the WEBQUESTIONS dataset. Researchers have approached this problem from different angles. Semantic parsing (Berant et al., 2013; Berant and Liang, 2014) aims to predict the logic forms of the question given the distant supervision of direct answers. Their logic forms were derived from dependency parses and then converted into database queries. Reddy et al. (2014) conceptualized semantic parsing as a graph matching problem by building graphs with Combinatory Categorial Grammar parses. Edges and nodes in parsing graphs were grounded with respect to Freebase relations and entities. Other research explored the graph nature of Freebase. For instance, Bordes et al. (2014) learned low-dimensional word embeddings for both the question and related topic subgraph. A scoring function was defined over these embeddings so that correct answers yielded a higher score. Yao and Van Durme (2014) treated this task as a direct information extraction problem: each entity node from a topic graph was ranked against others by searching a massively generated feature space.

All of the above work resorted to using the Freebase annotation of ClueWeb (Gabrilovich et al., 2013) to gain extra advantage of paraphrasing QA pairs or dealing with data sparsity problem. However, ClueWeb is proprietary data and costs hundreds of dollars to purchase. Moreover, even though the implementation systems from (Berant et al., 2013; Yao and Van Durme, 2014; Reddy et al., 2014) are open-source, they all take considerable disk space (in tens of gigabytes) and training time (in days). In this paper we present a system that can be easily implemented in 300 lines of Python code with no compromise in accuracy and speed.

3 Search

Given a question, we need to find out all named entities (or *topics* in Freebase terms). For instance, for the question what character did natalie portman play in star wars?, we are mainly interested in the topics of natalie portman and star wars. Note that all sentences in WEBQUESTIONS are lowercased.

Normal approaches require a combination of basic NLP processing. For instance, an NER tagger might recognize natalie portman as a PERSON, but would not recognize star wars as a movie, unless there is a pre-defined gazetteer. Then one needs to resort to basic chunking to at least identify star wars as a noun phrase. Moreover, these NLP tools need to be trained to better adapt lowercased sentences. Even though, one is still limited to a small number of recognizable types: noun phrases, person, location, organization, time, date, etc.

Freebase contains 46 million topics, each of which is annotated with one or more types. Thus a natural idea is to use these 46 million topics as a gazetteer and recognizes named entities from the question (with ambiguities), with two steps:

1. enumerate all adjacent words (of various length) of the question, an $O(N^2)$ operation where N is the length of question in words;

2. check whether each adjacent word block exists in the gazetteer.

We use two common data structures to search efficiently, with three design principles:

1. compact and in-memory, to avoid expensive hard disk or solid state drive I/O;

2. fuzzy matching, to be robust against noise;

3. easily extensible, to accommodate new topics.

3.1 Fuzzy Matching and Generation

To check whether one string is a Freebase topic, the easiest way is to use a hash set. However, this is not robust against noise unless a fuzzy matching hashing function (e.g., locality sensitive hashing) is designed. Moreover, 46 million keys in a giant hash set might cause serious problems of key collision or set resizing in some programming languages. Instead, we propose to use two common data structures for the purpose of fuzzy matching or generation.

Fuzzy Matching with Sorted List [1]: a sorted list can provide basic fuzzy matching while avoiding the key collision problem with slightly extra computing time. The search is done via 3 steps:

1. build a sorted list of 46 million topics;
2. to identify whether a string appears in the list, do a binary search. Since 46 million is between 2^{25} and 2^{26}, a search would require in the worst case 26 steps down the binary random access ladder, which is a trivial computation on modern CPUs;
3. For each string comparison during the binary search, also compute the edit distance. This checks whether there is a similar string within an edit distance of d in the list given another string.

Note that a sorted list does not compute *all* similar strings within an edit distance of d efficiently. Adjacent strings in the list also wastes space since they are highly similar. Thus we also propose to use a prefix tree:

Fuzzy Generation with Prefix Tree (Trie): a prefix tree builds a compact representation of all strings where common prefixes are shared towards the root of the tree. By careful back tracing, a prefix tree can also output all similar strings within a fixed edit distance to a given string. This efficiently solves the wasted space and generation problems.

3.2 Implementation and Speed Comparison

We maximally re-used existing software for robustness and quick implementation:

[1]We mix the notion of array vs. list as long as the actual implementation satisfies two conditions: O(1) random access time and O(1) appending(resizing) time.

	$\mathbf{d=0}$	$\mathbf{d=1}$	$\mathbf{d=2}$
Fuzzy Matching (Sorted List, PyPy)	<0.01ms	7.9ms	7.5ms
Fuzzy Generation (Trie, Elasticsearch)	29ms	210ms	1969ms

Table 1: Fuzzy query time per question. d is the edit distance while $d = 0$ means strict matching. HTTP roundtrip overhead from Elasticsearch was also counted.

Sorted List was implemented with vanilla Python list, compiled with the PyPy just-in-time compiler.

Prefix Tree was implemented with Elasticsearch, written in Java.

Both implementations held 46 million topic names (each topic name is 20 characters long on average) in memory. Specifically, sorted list took 2.06GB RAM while prefix tree took 1.62GB RAM.

Then we tested how fast it was to find out all topics from a question. To do this, we used the DEV set of WEBQUESTIONS. Enumerating all adjacent words of various length is an $O(N^2)$ operation where N is a sentence length. In practice we counted 27 adjacent words on average for one question, thus 27 queries per question. Elasticsearch follows the client-server model where client sends HTTP queries to the backend database server. To reduce HTTP roundtrip overhead, we queried the server in burst mode: client only sends one "mega" query to the server per question where each "mega" query contains 27 small queries on average. Experiments were conducted with an Intel Core i5-4278U CPU @ 2.60GHz.

Table 1 shows the query time per question. Note that this is an evaluation of real-world computing situation, not how efficiently either search structure was implemented (or in what programming language). Thus the purpose of comparison is to help choose the best implementation solution.

3.3 Ranking

After identifying all possible topic names in a question, we send them to the official Freebase Search API to rank them. For instance, for the question what character did natalie portman play in star wars?, possible named entities include character, natalie, natalie portman, play, star, and star wars.

But in general natalie portman and star wars should be ranked higher. Due to the crowd-sourced nature, many topics have duplicate entries in Freebase. For instance, we counted 20 different natalie portman's (each one has a unique machine ID), but only one is extensively edited. One can either locally rank them by the number of times each topic is cross-referenced with others, or use the Freebase Search API in an online fashion. In our experiments the latter yielded significantly better results. The Freebase Search API returns a ranked list of topic candidates. Our next job is to predict answers from this list.

4 Prediction

Given a question and its topic, we directly predict the relation that connects the topic with the answer. Our features and model are extremely simple: we took unigram and bigram words from the question as our features and used logistic regression to learn a model that associates lexical words with relations.

The training set of WEBQUESTIONS contains 3778 question and answer pairs. Each question is also annotated with the Freebase topic used to identify the answers. Then for each of the topic-answer pairs, we extracted a direct relation from the topic to the answer, for instance (TOPIC: Sweden, RELATION: */location/country/currency_used*, ANSWER: Swedish krona). If there were more than one relations between the topic and the answer (mostly due to dummy "compound" nodes in between), we chose the nearest one to the answer node as the direct relation. To be more specific: we first selected the the shortest path between the topic and answer node, then chose the relation from the answer node to its parent node, regardless of whether the parent node was the topic node. In this way we found direct relations for 3634 of these questions, which count as 96.2% of the whole training set.

Note that we "reverse-engineered" the slot-filling relations that would predict the correct answers based on annotated gold answers. It does not mean that these relations will predict the answers with 100% accuracy. For instance, for the question what was the first book dr. seuss wrote?, the direct relation was */book/author/book_editions_published*. However, this relation would predict *all* books Dr. Seuss wrote, instead of just the first one. Thus in the training set, we further counted the number of relations that point to the *exact* gold answers. In all, 62% of questions out of the whole training set can be exactly answered by a single relation.

The remaining 38% presented a complicated case. We sampled 100 questions and did a manual analysis. There were mainly two reasons that contributed to the 38%:

1. **Noisy Annotation**: questions with incomplete answers. For instance,

   (a) for the question what does bob dylan sing?, the annotated answer was only "like a rolling stone", while the direct relation */music/artist/track* gave a full list;

   (b) for the question what kind of currency does cuba use?, the annotated answer was Cuban Peso, while the direct relation */location/country/currency_used* led to two answers: Cuban Peso and Cuban Convertible Peso.

2. **Complex Questions**: questions with constraints that cannot be answered by binary relations. For instance:

   (a) who does david james play for 2011?

   (b) which province in canada is the most populated?

   (c) who does jodelle ferland play in eclipse?

For category 1, the answers provided by direct binary relations will only hurt evaluation scores, but not user experience. For category 2, we counted about 40% of them from the samples. Thus in total, complex questions constructed $38\% \times 40\% = 15\%$ of the whole training set. In other words, 85% of questions can be answered by predicting a single binary relation. This provides statistical evidence that the task of QA on WEBQUESTIONS can be effectively simplified to a tuple slot-filling task.

5 Results

We applied Liblinear (Fan et al., 2008) via its Scikit-learn Python interface (Pedregosa et al., 2011) to train the logistic regression model with L2 regularization. Testing on 2032 questions took 51 seconds.[2]

[2]This excluded the time used to call the Freebase Search API, which is highly dependent on the network and server node.

	$\mathbf{F_1}$ (Berant)	$\mathbf{F_1}$ (Yao)
Yao and Van Durme (2014)	33.0	42.0
Berant and Liang (2014)	39.9	43.0
Reddy et al. (2014)	41.3	-
Bordes et al. (2014)	41.8	45.7
this work	**44.3**	**53.5**

Table 2: Results on the WEBQUESTIONS test set.

We found no difference in quality but only slightly in speed in the search results between using sorted list and prefix tree. Moreover, in specifically the WEBQUESTIONS dataset, there was no difference in strict matching and fuzzy matching – the dataset is somehow void of typographical errors. [3]

We evaluated both the average F_1 over all questions (Berant) and the F_1 of average precision and recall values (Yao) following Bordes et al. (2014), shown in Table 2. Our method outperformed all previous systems in both F_1 measures, with possibly two reasons: 1, the simplicity of this method minimizes error propagation down the processing pipeline; 2, we used direct supervision while most previous work used distant supervision.

6 Limitation and Discussion

The limitation of our method comes from the assumption: most questions can be answered by predicting a direct binary relation. Thus it cannot handle complex questions that require to resolve a chain of relations. These complex questions appear about 15% of the time.

Note that WEBQUESTIONS is a realistic dataset: it was mined off Google Suggest, which reflects people's everyday searches. Our manual analysis showed that these complex questions usually only contain one type of constraint that comes from either a ranking/superlative describer (first, most, etc) or a preposition phrase (in 1998, in some movie, etc). To adapt to these questions, we can take a further step of learning to filter a returned list of results. For instance, first (first husband, first novel, etc) requires learning a time ordering; a prepositional constraint usually picks out a single result from a list of results. To go beyond to "crossword-like" questions with multiple constraints, more powerful mechanisms are certainly needed.

In summary, we have presented a system with a focus on efficiency and simplicity. Computation time is minimized to allow more time for network traffic, while still being able to respond in real time. The system is based on a simpler assumption: most questions can be answered by directly predicting a binary relation from the question topic to the answer. The assumption is supported by both statistics and observation. From this simple but verified assumption we gained performance advantages of not only speed, but also accuracy: the system achieved the best result so far on this task.

[3] This is likely due to the fact that the dataset was crawled with the Google Suggest API, which aggregates common queries and common queries are mostly free of typos. For real-world everyday queries, fuzzy matching should still be applied.

References

Jonathan Berant and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of ACL*.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of EMNLP*.

Antoine Bordes, Sumit Chopra, and Jason Weston. 2014. Question answering with subgraph embeddings. In *Proceedings of EMNLP 2014*, pages 615–620.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874.

Evgeniy Gabrilovich, Michael Ringgaard, , and Amarnag Subramanya. 2013. FACC1: Freebase annotation of ClueWeb corpora. http://lemurproject.org/clueweb09/FACC1/.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *The Journal of Machine Learning Research*, 12:2825–2830.

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

Xuchen Yao and Benjamin Van Durme. 2014. Information extraction over structured data: Question answering with freebase. In *Proceedings of ACL*.