OPEN-DOMAIN QUESTION ANSWERING WITH PRE-CONSTRUCTED QUESTION SPACES

Jinfeng Xiao
University of Illinois at Urbana-Champaign
jxiao13@illinois.edu

Lidan Wang
Adobe Inc.
lidwang@adobe.com

Franck Dernoncourt
Adobe Inc.
dernonco@adobe.com

Trung Bui
Adobe Inc.
bui@adobe.com

Tong Sun
Adobe Inc.
tsun@adobe.com

Jiawei Han
University of Illinois at Urbana-Champaign
hanj@illinois.edu

ABSTRACT

Open-domain question answering aims at solving the task of locating the answers to user-generated questions in large collections of documents. There are two families of solutions to this challenge. One family of algorithms, namely retriever-readers, first retrieves some pieces of text that are probably relevant to the question, and then feeds the retrieved text to a neural network to get the answer. Another line of work first constructs some knowledge graphs from the corpus, and queries the graph for the answer. We propose a novel algorithm with a reader-retriever structure that differs from both families. Our algorithm first reads off-line the corpus to generate collections of all answerable questions associated with their answers, and then queries the pre-constructed question spaces online to find answers that are most likely to be asked in the given way. The final answer returned to the user is decided with an accept-or-reject mechanism that combines multiple candidate answers by comparing the level of agreement between the retriever-reader and reader-retriever results. We claim that our algorithm solves some bottlenecks in existing work, and demonstrate that it achieves superior accuracy on a public dataset.

Keywords question answering, open-domain question answering, question space, reader-retriever

1 Introduction

Open-domain question answering, abbreviated as *OpenQA* in this paper, aims at enabling computers to answer user-submitted questions based on a large collection of documents (i.e. a corpus). It has wide applications in document understanding, user interaction, and decision making. While the research community and data-driven companies have long been pushing forward the cutting-edge research and industrialization of question answering methods, there remains large room for improvement, especially under the open-domain setting.

There are two families of state-of-the-art algorithms to solve OpenQA tasks. One family, namely retriever-readers (Fig. 1, left branch), first retrieves from the corpus some much shorter pieces of text that are likely to be relevant to the question, and then uses neural networks to read the retrieved passages and locate the answer. Another line of work, namely question answering using knowledge bases (or QA using KB; Fig. 1, middle branch), first constructs a knowledge base (KB) from the corpus, then decomposes the questions into events, and finally queries the KB for the answer. The knowledge base contains one or more knowledge graphs, which typically include typed entities and their relations. Either family of algorithms has some pros and cons to be discussed in later sections.

We propose a novel reader-retriever algorithm for OpenQA. The design, as shown with the right branch in Fig. 1, differs from any existing work. First, we use deep neural networks to read the corpus offline, detect named entities, generate questions, and collect the results into two spaces of all questions answerable with the corpus. When users enter queries online, a retriever compares user queries with the pre-constructed question spaces to retrieve the answers that are most likely to be asked in the given way. We then aggregate the resulted answers with the retriever-reader-generated answers,
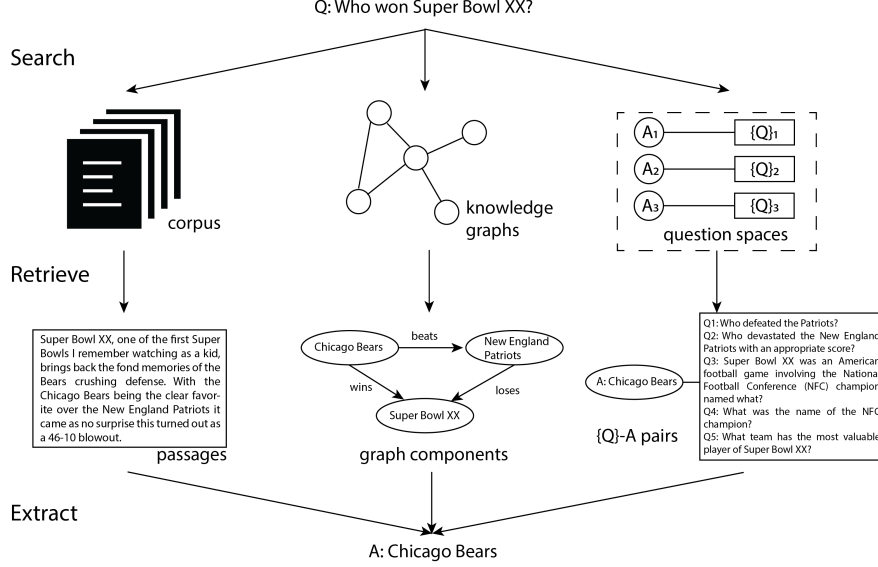
Figure 1: Retriever-readers (left), QA using KB (middle), and reader-retrievers (right).

using an accept-or-reject mechanism that combines multiple candidate answers by comparing the level of agreement between them. Experiments on a public dataset show that proper use of those pre-constructed question spaces can boost the performance of OpenQA.

Different from the previous retriever-reader models where for each query only a tiny portion of the corpus is fed into a deep neural network, our method ensures that every query benefits from a deep scan over the entire corpus so that information throughout the corpus can be aggregated to contribute to the final answer. Compared to QA using KB methods, our method can succeed as long as the answer entity is correctly detected, and doesn't require correct identification of all the three elements in a <head entity, relation, tail entity> triplet. Our main contributions include:

- We propose a novel reader-retriever algorithm for OpenQA, where we first construct offline two spaces of all answerable questions from the corpus, and then retrieve online the answers that are most likely to be asked in the given way.
- We propose an accept-or-reject mechanism to combine answers returned by a retriever-reader and two reader-retrievers with different question spaces.
- We show on a public dataset that the above procedure gives state-of-the-art OpenQA performance.

2 Related Work

2.1 Open-Domain Question Answering

Open-Domain Question Answering (OpenQA) aims at answering questions based on large collections of documents. It is harder than standard question answering (QA), where the task is to find the answers from short text passages. While recent deep learning models have achieved over 85% accuracy on standard QA [23], those models cannot be directly applied to OpenQA due to two constraints. First, it is computationally expensive and unnecessary to scan the entire corpus with a deep model for answering a specific question, whose answer is usually contained in just a few sentences. Second, the large body of irrelevant text will make the signal-to-noise ratio really small and thus result in bad accuracy, even if the entire corpus can be directly fed into a deep model. Thus, two new families of algorithms have been designed to solve OpenQA challenges: 1) retriever-readers, and 2) QA using KB.

2.2 Retriever-Readers

It is natural to solve OpenQA by converting it to standard QA with information retrieval (IR) techniques, and that gives the popular retriever-reader algorithmic design. One well-known example is DrQA [4], which answers open-domain questions by first retrieving relevant Wikipedia pages with TF-IDF scores [17], then reading the question and the retrieved page with recurrent neural networks (RNNs) [9, 3], and finally predicting the answer span by computing the similarity between the RNN-generated representations of the query and the retrieved paragraphs.

Unfortunately, although adding a retriever before a reader converts OpenQA to QA and thus makes existing QA methods readily applicable to OpenQA tasks, such a solution is far from perfect. When the retriever module is computationally efficient, the retrieved results will not be very reliable, and the performance of the subsequent reader module is constrained by the rather poor accuracy of the retriever [11]. For example, the exact match accuracy of DrQA for standard QA on the SQuAD dataset [16] is as good as 69.5%, but it dramatically decreases to 28.4% on the same dataset under open-domain settings [4]. Another disadvantage of computationally efficient retrievers is that they lack trainable parameters for recovering from mistakes [5]. On the other hand, there exist systems such as R^3 [20] and DS-QA [15] that have sophisticated retrievers jointly trained with the readers, but such designs are not scalable to large corpora [5].

2.3 QA Using KB

There are solutions that solve OpenQA with knowledge bases. Such approaches involve an offline knowledge graph construction module and an online graph query module. The graph construction module scans the entire corpus, detects entities, recognizes entity relations, and organizes those relations into a knowledge base that contains one or more knowledge graphs. Once a knowledge base is constructed, OpenQA tasks can then be converted to graph search tasks: finding the most likely answer is done by searching the most relevant node or edge in the graph. The search can be done in various ways, for example by template decomposition [24] or graph embedding [12]. When a user submits a query question, the engine first decomposes the natural language query into events, and extracts relevant entities from the graph as the answer. Examples of QA using KB applications include Google Knowledge Graph and Bing Satori [19].

Despite the success of those applications, a lot of challenges remain, and the limitations of currently available graph-based approaches are also apparent. It is a common practice to decompose a natural language question into a triplet of <head entity, relation, tail entity> so that a graph query can be run. This confines QA using KB to short factoid questions with easy syntax, and thus better query decomposition remains an active research topic. Due to the fact that the search space is constrained by the graph, a graph-based OpenQA method can only answer questions that are fully covered by the graph, and errors in the entity detection and relation extraction modules in the offline graph construction stage will propagate to the online query stage. Unlike in the retriever-reader world where some deep learning methods like BERT [6], XLNet [21] and their variants are clear winners, there are no widely accepted winners in the QA using KB family, and in many cases which QA using KB method works better depends on the actual corpus and task at hand.

2.4 Question Generation

Another topic relevant to our algorithm is question generation (QG). The task is to generate a question whose answer is a given text span in a given passage. Most recent approaches train encoder-decoders on public QA datasets like SQuAD so that the generated questions mimic the actual questions in the training set [14, 25, 22].

3 Approach

To solve the challenges faced by existing work as discussed in Section 2, we propose a reader-retriever model structure and an accept-or-reject mechanism for answer aggregation. Before making any online responses, the entire corpus is scanned offline by a deep learning reader to generate all answerable questions and collect them into two question spaces. After that, our algorithm responds to online queries by retrieving the questions that best matches the user query, and use an accept-or-reject mechanism to decide whether to trust the answers to the retrieved questions or to accept a retriever-reader-generated answer. In this way, the power of the deep learning reader is no longer restricted by the accuracy of the retriever as what happens in retriever-reader models, and the question spaces are more flexible and friendly to natural language queries than knowledge graphs are.

3.1 Question Spaces

Definition 1. A *question space* is a bipartite graph with two disjoint and independent node sets A and Q representing the answers and associated questions. There are two types of question spaces: QA Spaces and {Q}A Spaces. In a *QA Space*, each element $a_{i,j}$ of A represents the jth occurrence in the corpus of the ith distinct named entity, and each element $q_{i,j}$ of Q is a question generated from the context of $a_{i,j}$ with a_i as its answer. For every i and j, $a_{i,j}$ and $q_{i,j}$ form a *QA pair* and are connected in the graph. In a *{Q}A Space* (pronounced as Q-set-A-space), each element a_i of A represents the ith distinct named entity, and each element q_i of Q is a collection of the $q_{i,j}$'s for all j in the QA Space. For every i, a_i and q_i form a *{Q}A pair* and are connected in the graph. In short, a QA space contains pairs of answers and generated questions, while a {Q}A space contains pairs of distinct answers and collections of all generated questions with that answer.
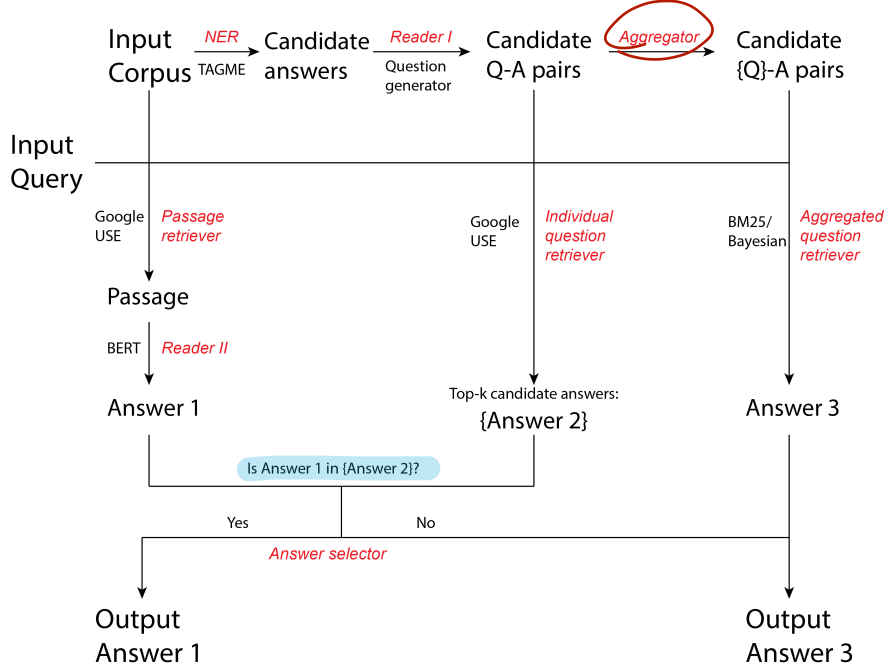
Figure 2: Detailed structure of the proposed method.

For example, given the five questions in Fig. 1 whose answer is "Chicago Bears", the QA Space will have five QA pairs: $\{a_{1,1}$ = "Chicago Bears", $q_{1,1}$ = "Who defeated the Patriots?"$\}$, ..., $\{a_{1,5}$ = "Chicago Bears", $q_{1,5}$ = "What team has the most valuable player of Super Bowl XX?"$\}$, and the {Q}A space will have one {Q}A pair: $\{a_1$ = "Chicago Bears", q_1 = {"Who defeated the Patriots?", ..., "What team has the most valuable player of Super Bowl XX?"}$\}$.

3.2 Detailed Structure

The detailed structure of our method is given in Figure 2. The first row in the graph shows the offline reader component that generates two question spaces namely a QA Space and a {Q}A Space, and the modules below the input query are all online. The online part includes three retrievers, another reader, and an answer selector.

3.2.1 NER, Reader I and Aggregator

Given a corpus, a named entity recognition tool called TAGME [7, 8] is applied to detect named entities from the corpus and link the entities to Wikipedia pages. Those entities form the set of candidate answers A in Definition 1, Section 3.1. Then a question generator is applied to the set of candidate answers to generate a question for each answer. The question generator features an encoder-decoder structure with a question answering reward and a question fluency reward tuned with policy gradient optimization [22, 10]. This finishes the construction of the QA Space. We then use an aggregator to build the {Q}A Space by putting together all the questions with the same answer.

3.2.2 Passage Retriever and Reader II

Given a query, the passage retriever uses the dot product of the query embedding and passage embedding vectors generated by Google Universal Sentence Encoder (Google USE) [2] to retrieve from the corpus a passage that is semantically most similar to the query. We then use BERT [6], fine-tuned on SQuAD, to read the retrieved passage, predict the answer, and record the predicted answer as *Answer 1*. We denote this baseline approach as **BERT-large** or **BERT-base**, depending on which BERT model is used.

3.2.3 Individual Question Retriever

Given a query, the individual question retriever uses Google USE to retrieve from the QA space k questions that are semantically most similar to the query. We denote this approach as **QA-space-only**, and record the set of answers associated with the k retrieved questions as *{Answer 2}*.

3.2.4 Aggregated Question Retriever

Given a query, the aggregated question retriever uses the BM25 score [18] and the Bayesian maximum posterior probability [1] to retrieve from the {Q}A space the answer whose associated set of questions is most similar to the given query. We denote this approach as **{Q}A-space-only**, and record the retrieved answer as *Answer 3*.

3.2.5 Answer Selector

The last step is to select which answer to return to the user. Our answer selector works as follows: If Answer 1 appears in the set {Answer 2}, then accept Answer 1 and return it; otherwise reject Answer 1 and return Answer 3. The intuition is that we accept the baseline prediction if it agrees with the result from the QA space, and otherwise decline the baseline prediction and return an answer based on the {Q}A space. We denote the complete workflow depicted in Figure 2 as **reader-retriever-full**.

4 Evaluation

In this section, we evaluate the accuracy of five methods, namely reader-retriever-full, BERT-large, BERT-base, QA-space-only, and {Q}A-space-only, as described in Section 3.2. To evaluate QA-space-only, we set $k = 10$ and take the answer with the highest weighted vote among the top k candidate answers as the predicted answer. The top 1 answer contributes 10 votes, top 2 contributes 9 votes, and so on.

We use the TriviaQA public dataset [13], which contains over 650K question-answer-evidence triples and on average each question-answer pair has six evidence documents. While the leaderboard evaluation allows algorithms to know in advance which six documents support which question-answer pair, we use more realistic open-domain settings in our experiments. We mix all documents together to form our corpus, name it *TriviaQA-Open*, and for each question we ask the algorithms to identify the answer from this corpus. In addition, to mimic the realistic scenario where algorithms seldom know exactly how users actually behave at its training time, we do not allow algorithms to see any portion of TriviaQA during training. Thus all algorithms are trained on SQuAD [16], a dataset containing 107K question-answer pairs from 536 articles.

Table 1: Test accuracy on TriviaQA-Open.

Method	Accuracy
reader-retriever-full	**0.30**
BERT-large	0.16
BERT-base	0.15
QA-space-only	0.07
{Q}A-space-only	0.21

Test accuracy on TriviaQA-Open of the five methods is shown in Table 1. The numbers indicate that the OpenQA task is hard, which is consistent with our discussion in Section 2. The baseline retriever-reader models achieve about 16% accuracy, and running the resource-consuming BERT-large does not contribute much on top of the smaller and faster BERT-base. Comparing {Q}A-space-only versus BERT-large implies that taking the results obtained from the {Q}A Space looks like a better choice as its accuracy is 5% higher than the baselines. Although QA-space-only does not seem to work by itself, its role in deciding whether to accept or reject the baseline result is clearly beneficial. Thanks to the contribution from the two question spaces and the accept-or-reject answer selection mechanism, our complete model reader-retriever-full outperforms the BERT baselines by 14% and our {Q}A-space-only method by 9%. Those results indicates that reader-retrievers with question spaces generate state-of-the-art OpenQA accuracy, and integrating them with retriever-readers further boosts the performance in a significant manner.

5 Case Study: Why It Works?

Now let's refer back to the example in Figure 1 to get more intuition about why question spaces work. The fact is that the question "who won Super Bowl XX" is a real question from TriviaQA, the right branch of Figure 1 is the only real result observed in experiments, and the left and middle branches show some ideal results that do not actually happen. In reality, the left branch retrieves an irrelevant passage and thus gets a wrong answer, while the middle branch also fails because our entity recognition module fails to recognize the entity "Super Bowl XX". On the other hand, it does recognizes the entity "Chicago Bears", and the question space does contain clues that connect the correct answer to the

query. Although there is no question in the QA Space that exactly matches the given query, by aggregating the signal from multiple questions in the {Q}A Space, our algorithm succeeds in finding the answer.

6 Conclusion and Future Work

We propose a novel algorithm of constructing question spaces from corpora and using them to improve open-domain question answering. Results on a public dataset show that the proposed structure outperforms the retriever-reader BERT baseline by a margin of 14%.

As future plans, we are interested in fully unleash the potential of reader-retriever models by making the components stronger. For the reader component, we aim at reducing the false positive QA pairs generated from the corpus, and for the retriever, we plan to study how to better capture the signal in the question spaces.

References

[1] BOX, G. E., AND TIAO, G. C. *Bayesian inference in statistical analysis*, vol. 40. John Wiley & Sons, 2011.

[2] CER, D., YANG, Y., KONG, S.-Y., HUA, N., LIMTIACO, N., JOHN, R. S., CONSTANT, N., GUAJARDO-CESPEDES, M., YUAN, S., TAR, C., ET AL. Universal sentence encoder. *arXiv preprint arXiv:1803.11175* (2018).

[3] CHEN, D., BOLTON, J., AND MANNING, C. D. A thorough examination of the cnn/daily mail reading comprehension task. *arXiv preprint arXiv:1606.02858* (2016).

[4] CHEN, D., FISCH, A., WESTON, J., AND BORDES, A. Reading wikipedia to answer open-domain questions. *arXiv preprint arXiv:1704.00051* (2017).

[5] DAS, R., DHULIAWALA, S., ZAHEER, M., AND MCCALLUM, A. Multi-step retriever-reader interaction for scalable open-domain question answering. *arXiv preprint arXiv:1905.05733* (2019).

[6] DEVLIN, J., CHANG, M.-W., LEE, K., AND TOUTANOVA, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[7] FERRAGINA, P., AND SCAIELLA, U. Tagme: on-the-fly annotation of short text fragments (by wikipedia entities). In *Proceedings of the 19th ACM international conference on Information and knowledge management* (2010), ACM, pp. 1625–1628.

[8] FERRAGINA, P., AND SCAIELLA, U. Fast and accurate annotation of short texts with wikipedia pages. *IEEE software 29*, 1 (2011), 70–75.

[9] HERMANN, K. M., KOCISKY, T., GREFENSTETTE, E., ESPEHOLT, L., KAY, W., SULEYMAN, M., AND BLUNSOM, P. Teaching machines to read and comprehend. In *Advances in neural information processing systems* (2015), pp. 1693–1701.

[10] HOSKING, T., AND RIEDEL, S. Evaluating rewards for question generation models. *arXiv preprint arXiv:1902.11049* (2019).

[11] HTUT, P. M., BOWMAN, S. R., AND CHO, K. Training a ranking function for open-domain question answering. *arXiv preprint arXiv:1804.04264* (2018).

[12] HUANG, X., ZHANG, J., LI, D., AND LI, P. Knowledge graph embedding based question answering. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining* (2019), ACM, pp. 105–113.

[13] JOSHI, M., CHOI, E., WELD, D. S., AND ZETTLEMOYER, L. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *arXiv preprint arXiv:1705.03551* (2017).

[14] KIM, Y., LEE, H., SHIN, J., AND JUNG, K. Improving neural question generation using answer separation. In *Proceedings of the AAAI Conference on Artificial Intelligence* (2019), vol. 33, pp. 6602–6609.

[15] LIN, Y., JI, H., LIU, Z., AND SUN, M. Denoising distantly supervised open-domain question answering. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2018), pp. 1736–1745.

[16] RAJPURKAR, P., ZHANG, J., LOPYREV, K., AND LIANG, P. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[17] RAMOS, J., ET AL. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning* (2003), vol. 242, Piscataway, NJ, pp. 133–142.

[18] ROBERTSON, S., ZARAGOZA, H., ET AL. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval 3*, 4 (2009), 333–389.

[19] UYAR, A., AND ALIYU, F. M. Evaluating search features of google knowledge graph and bing satori: entity types, list searches and query interfaces. *Online Information Review 39*, 2 (2015), 197–213.

[20] WANG, S., YU, M., GUO, X., WANG, Z., KLINGER, T., ZHANG, W., CHANG, S., TESAURO, G., ZHOU, B., AND JIANG, J. R 3: Reinforced ranker-reader for open-domain question answering. In *Thirty-Second AAAI Conference on Artificial Intelligence* (2018).

[21] YANG, Z., DAI, Z., YANG, Y., CARBONELL, J., SALAKHUTDINOV, R. R., AND LE, Q. V. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems* (2019), pp. 5754–5764.

[22] YUAN, X., WANG, T., GULCEHRE, C., SORDONI, A., BACHMAN, P., SUBRAMANIAN, S., ZHANG, S., AND TRISCHLER, A. Machine comprehension by text-to-text neural question generation. *arXiv preprint arXiv:1705.02012* (2017).

[23] ZHANG, Z., YANG, J., AND ZHAO, H. Retrospective reader for machine reading comprehension. *arXiv preprint arXiv:2001.09694* (2020).

[24] ZHENG, W., YU, J. X., ZOU, L., AND CHENG, H. Question answering over knowledge graphs: question understanding via template decomposition. *Proceedings of the VLDB Endowment 11*, 11 (2018), 1373–1386.

[25] ZHOU, Q., YANG, N., WEI, F., TAN, C., BAO, H., AND ZHOU, M. Neural question generation from text: A preliminary study. In *National CCF Conference on Natural Language Processing and Chinese Computing* (2017), Springer, pp. 662–671.