Open-Domain Question Answering

Abstract

In recent years, researchers have obtained state-of-art results in close-domain machine comprehension. On the other hand, the open-domain machine comprehension has a much worse performance compared to the open-domain one. Seeing the potential for improvements, in this paper we propose a novel solution to handle open-domain question answering by using Wikipedia as source reference. Our model consists of a lucene based search engine to retrieve passages from Wikipedia, a ranker that assigns scores to different passages based on their likelihoods to contain the answer, and a reader that extracts the answer as a text-span from the top-scored passages. Our experimental result shows that our model is competitive, and future improvements could lead to state-of-art result.

1 Introduction

The first thing we consider is our data source for question answering. In order to obtain a general model that could answer various topics of questions, we need an enormous amount of text data as our resource to search for the result, so we utilize Wikipedia article collections as our knowledge source. Wikipedia collection provides over 5 million articles with diverse topics, which is sufficient to cover the topics appearing in most of open-domain questions. Moreover, Wikipedia is a constantly growing community with new or updated materials everyday, so it will cover the topics that may happen in the future questions.

With all 5 million articles as our basis, we cannot simply go over the whole data pool and search for the answer in every article. We then introduce several different components to help split the big problem into three subproblems. Correspondingly, we propose an end-to-end framework consisting of three models: a search engine, which is fed with the question and outputs a list of paragraphs that are most relevant to the question, a ranker, which ranks the list of paragraphs from search engine by the likelihood of the context to contain the answer, and a reader, which predicts the starting index and the ending index of the answer given the top-ranked paragraphs and returns the best answer span. We will discuss the retriever-ranker-reader model in detail in section 3.

We also try out different experiments on different parts of the project. From the start of project, we experiment different levels of indexing of the context for search engine, including paragraph level indexing and article level indexing. At paragraph level indexing, we try bi-gram indexing and uni-gram indexing. We also experiment whether using sentence embedding to rank test would outperform traditional TF-IDF and expensive attention based approaches. We use Infersent (Conneau et al., 2017), a sentence embeddings method that provides semantic sentence representations, as our sentence encoding method. We also set up the experiment to evaluate the performances of different combinations of components in the model, and present these performances in the end.

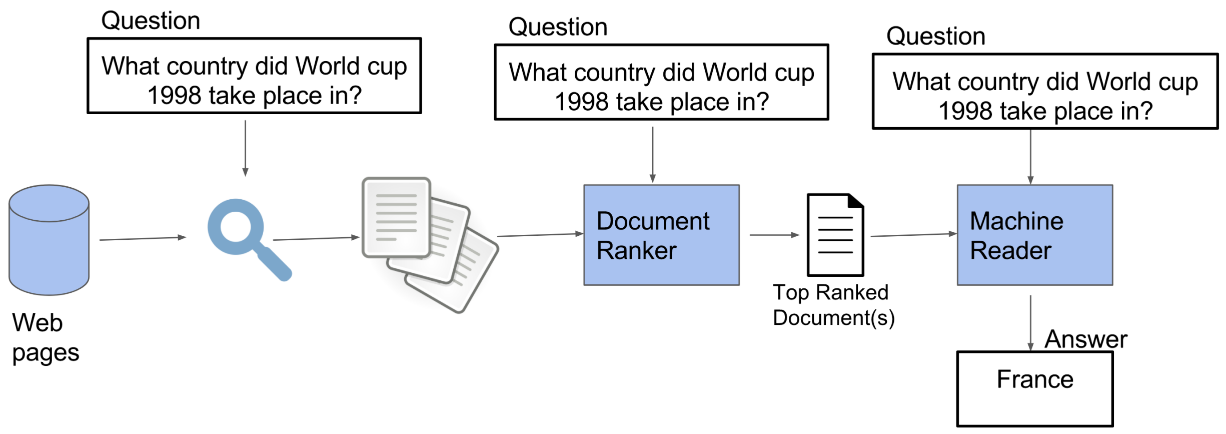
Question
What country did World cup 1998 take place in?
Question
What country did World cup 1998 take place in?
Question
What country did World cup 1998 take place in?
Web pages
Document Ranker
Top Ranked Document(s)
Machine Reader
Answer
France

Figure 1: The overall architecture of the model

2 Related Work

Open-domain question answering is defined as retrieving the answers of domain-unspecific questions from a collections of referencing articles. The challenging part of open-domain QA is to identify the passage that is most relevant to the random-topic query, at the same time, the model needs to understand the relationship without learning the implicit connections of any domain-specific knowledge. Together these require the model to take a more general approach to detect the answer from the article collections. Open-domain QA has recently been a popular topics and has gained heavy attentions from NLP professionals and researchers. Our model is motivated from several research papers that already present state-of-art result on question answering and machine comprehension. (Chen et al., 2017) train Dr.QA, a model which reads Wikipedia to answer open-domain questions. The model builds a document retriever to obtain articles, and use a multi-layer recurrent neural network to work as the machine reader to detect answers from Wikipedia paragraphs. (Wang et al., 2017) build a reinforced rank-reader to obtain answers to open-domain questions. Their model consists of a ranker that sort the retrieved passages by how likely a certain passage contains the answer, and an answer extractor based on reinforcement learning. The reader generates rewards according to the similarity between ground truth answer and predicted answer, and then passes the reward to ranker to help the ranker learn better policies at each states. (Seo et al., 2017) build a machine reader which answers a query from a given context paragraph. The model introduces the bidirectional attention mechanism which utilizes context-to-query and query-to-context representations to summarize the relationship between query and context, leading to a state-of-art result in close-domain machine comprehension. Inspired by different parts of above works, we innovate our own model to handle open-domain question answering problem.

3 Model Architecture

As mentioned in introduction, we try several kinds of experiments. Specifically, the first one is to find the answer to a question given the article which contains the ground truth answer. The second experiment is to search for related passages to a given question, using search engine. Following (Wang et al., 2017), the overall architecture of our second approach consists of a search engine, a document ranker and machine reader. Similar document ranker and machine reader are used in the first approach.

3.1 Search Engine

We use PyLucene, a Python extension of Apache Lucene which is a popular open-source information retrieval software library, to index Wikipedia articles and build our search engine.

3.2 Passage Representation using Infersent

One of the objectives of our experiment is to explore whether the continuous representation of sen-

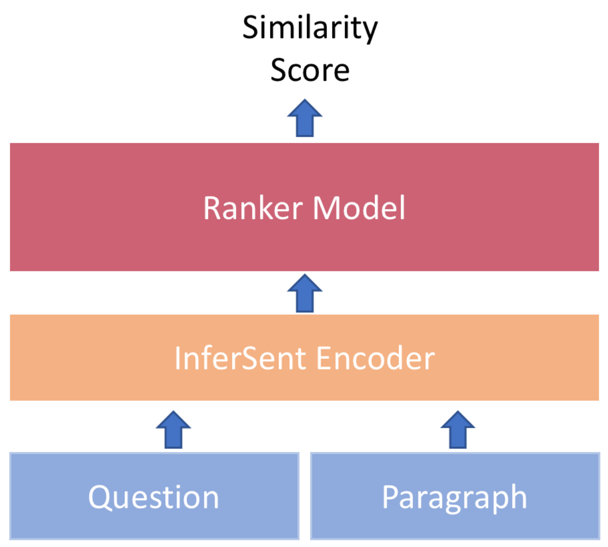

Figure 2: The Ranker Model

tences that can encode some semantic understanding of the sentences will perform well in the retrieval task. Therefore, we use Infersent to represent each sentence in the passage. To encode the whole paragraph, we simply sum the Infersent Embeddings of all the sentences in the passage. This approach is inspired by the sum of word embeddings as bags as words for sentence representation.

3.3 Ranker

Given a question and a paragraph, the ranker model acts as a scoring function that calculates the similarity between the question and the paragraph, as shown in Figure 2. In our experiment, we use a Bilinear model and a feedforward neural network as our ranker. However, any other scoring functions, for example, the attention based models, can also be used in this place. We train the ranker by minimizing the margin ranking loss (Collobert et al., 2011):

$$\sum_{p_-} max(0, 1 - f_+(q, p_+) + f_-(q, p_-)) \qquad (1)$$

where f is the score produced by ranker. Given a question and a list of paragraphs, the ranker will return the similarity scores between the question and each paragraph.

3.3.1 Bilinear Ranker

We use the same Bilinear model proposed by (Bai et al., 2009) as our ranker:

$$f(q, p) = [q : 1] * W * p \qquad (2)$$

3.3.2 Neural Network Ranker

We also implement a feedforward neural network as a scoring function. Following (Mou et al., 2016; Bowman et al., 2016), the input feature vector to the neural network is constructed by concatenating the question embedding, paragraph embedding, their difference, and an element-wise product:

$$\vec{x}_{classifier} = \begin{bmatrix} \vec{q} \\ \vec{p} \\ \vec{q} - \vec{p} \\ \vec{q} \bigodot \vec{p} \end{bmatrix} \qquad (3)$$

The neural network consists of a linear layer, followed by ReLu activation layer, and another linear layer. The output of the final linear layer is used as similarity score between question and paragraph.

3.4 Machine Reading

We use and compare BiDAF and DrQA for machine reading on the ranker results. For BiDAF, we use the pre-trained BiDAF model from AllenNLP API. We train the DrQA model from scratch by using the DrQA code available on Github.

BiDAF uses both word embedding and character embedding at the embedding layer. The two embeddings are stacked as one vector, which is further fed into Bidirectional LSTM to generate corresponding hidden states. The model utilizes these hidden states to compute context-to-query and query-to-context attentions. And these attentions are further fed into another layer of bidirectional LSTM to predict the starting index and the ending index of the answer span. (Seo et al., 2017)

DrQA first passes the paragraph and the question into two different multilayer Bidirectional LSTM to get paragraph encoding and question encoding. It then uses a bilinear term to capture the similarity between the paragraph vector and the question vector, and finally computes the probability of each token as being start and end of the answer span. (Chen et al., 2017)

3.5 Paragraph Selection

The ranker provides the similarity score between a question and each paragraph in the article. We select the top 5 paragraphs based on the scores provided by the ranker. We use softmax over the top 5 scores to find P(paragraph), the probability of each paragraph containing the ground truth.

Furthermore, Machine Readers provide the probability of each answer span given a paragraph **P(answer|paragaph)**. Using bayesian rule, we can thus calculate the probabilities of each answer span **P(paragraph, answer)** by multiplying probability of paragraph with the output probabilities from machine reader. We then choose the answer span with the highest P(answer, paragraph) as the output of our model.

4 Implementation Details

We use 2016-12-21 dump of English Wikipedia to index our search engine. We split each article into paragraphs and create the paragraph level index using Pylucene. We implement the search engine using Bigram index and BM25 ranking. We also experiment with Unigram index and find that Bigram index significantly improves the recall of the search engine. We also improve the recall of our search engine by appending the article's title at the beginning of each paragraph. This helps to add some sense of context which might be missing in the paragraph. We also try removing stop words in the index, but turns out that it doesnt improve the recall of search engine. We thus decide to include stop words in index.

Each Infersent embedding has 4096 dimension. Therefore, the input feature vector to our Neural Network ranker has 16384 dimension. The dimension of first linear layer of our Neural Network is 500, and the dimension of the second linear layer is 1.

We train both the Bilinear Ranker and the Neural Network Ranker using Stochastic Gradient with Adam optimizer. We train both of the models twice, one for ranking the paragraphs in the article and the other for ranking the paragraphs returned by search engine.

5 Evaluation

5.1 Search Engine

We can see from table that search engine with bigram index have significantly better recall than search engine with unigram index. Moreover, appending title to each paragraph helps as some paragraph may not contain the important named entities or keywords that are important for retrieval. However, as we can see from Table 1, the improvement in recall for appending title is not very large.

5.2 Ranker Evaluation

We find the Neural Network ranker performs significantly better than Bilinear Ranker during our initial evaluation. During the first stage of our experiment, which is building a ranker for paragraphs within the ground truth article, we find that Bilinear Ranker achieves only 36.85 recall at 1 while Neural Network model achieves 51.09 recall at 1. Therefore, we choose to use Neural Network ranker as our default ranker for subsequent experiments. The recall percentages of our Neural Network ranker is described in Table 2.

5.3 Machine Reading

We train both BiDAF model and DrQA model from scratch on SQuAD training set and tested them on the SQuAD validation set to make sure that they achieves the accuracy mentioned in the papers. Then we use the model to predict the answer given the result from previous part.

5.4 Article Level Ranker

The number of paragraphs in each article can range from 5 to 150. The average number of paragraphs in each article is 43. Given a question, the ranker will score each paragraph. The performance of machine reader on paragraph with the highest score is as described in Table 3 below. We can see that DrQA reader has exact match percentage of 33.62. Additionally, we can see the exact match percentage jumps to 40.8 if we do machine reading on the top 5 paragraphs and select the answer span with the highest P(paragraph, answer). However, we think that the exact match performance will be higher if we can use a better answer selection method. As P(paragraph, answer) is calculated

Table 1: Recall of Search Engines with different Paragraph level Indexes. Bigram index performs better than Unigram index. In addition, when title of each article is appended to the beginning of paragraph, the performance improves by a small amount.

	Bigram index (with title)	Bigram index (without title)	Unigram index (without title)
Recall@3	47.98	47.84	36.48
Recall@10	61.47	61.21	50.44
Recall@50	76.01	75.14	66.92
Recall@100	80.29	79.88	73.07
Recall@500	87.91	87.72	83.41

Recall@1	51.09
Recall@3	72.27
Recall@5	74.52

Table 2: Recall of Neural Network Ranker on paragraph level ranking

	EM	**F-1**
BiDAF(Top 1)	32.45	32.70
DrQA(Top 1)	33.62	40.14
DrQA(Top 5)	40.80	47.26
DrQA(Chen et. al)	49.40	-

Table 3: The Performance of reader after paragraph selection. The performance improves significantly when top 5 paragraphs are used for answering question

based on P(paragraph) and P(answer—paragraph), these two probabilities may not be the most accurate as both our ranker and reader are not perfect. To confirm this, we test the highest exact match we can get using the top 5 articles and find that its 49.96 for DrQA and 47.45 for BiDAF.

5.5 Combine Search Engine and Ranker

We also test our ranker and reader using the documents retrieved by the search engine. We retrieve 100 paragraphs from search engine and rank them using the neural network ranker which is pre-trained for paragraph selection. We then pass the top 5 paragraphs returned by the ranker to DrQA. The recall at top 5 paragraphs of the ranker is 43.5 when top 100 paragraphs are used, and 46.32 when top 50 paragraphs are used. The exact match and F1 score of the DrQA reader on the top 5 paragraphs(top 5 results of ranker on top 100 paragraphs retrieved from search engine) provided by ranker is 16.67 while the F1 score is 22.35. The reason for lower performance on the 100 paragraphs could be because our ranker is trained for paragraph selection within the article and the average number of paragraphs in each article in only 43. The performance of the ranker can be improved if we re-train on the results of our search engine.

	EM	**F-1**
BiDAF	47.45	62.21
DrQA	49.96	59.95
DrQA(Chen et. al, 2017)	49.40	-

Table 4: Upper bound performance achievable by the reader using top 5 paragraphs. We can see that it is possible to achieve comparable result as Chen et. al if we can use a better answer selection metric.

6 Conclusion and Future Work

We study the problem of open-domain question answering, by using Wikipedia as our source data, ranking the paragraphs from search result, and extracting the answer from top-ranked paragraphs. Our result shows that the most challenging part of our model is to get the best performance of each component and combine them together. We realize Infersent does fairly well, but using fixed size vectors doesnt seem to be the best. And we probably should use attention based architecture to get more competitive result. Moreover, training the ranker and reader together with reinforcement learning might help as it will help two components to adapt and optimize together. Finally we can improve the reader to extract the answers from multiple top weighted paragraphs so that it will find the best solution with prior knowledge.

References

Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Olivier Chapelle, and Killian Weinberger. 2009. Learning to rank with (a lot of) word features.

Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, and Christopher Potts. 2016. A fast unified model for parsing and sentence understanding. *arXiv*, (preprint arXiv:1603.06021).

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. *arXiv*, (preprint arXiv: 1704.00051).

Ronan Collobert, Jason Weston, Lon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, (11):2493.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv*, (preprint arXiv:1705.02364).

Lili Mou, Rui Men, Ge Li, Yan Xu, Lu Zhang, Rui Yan, and Zhi Jin. 2016. Natural language inference by tree-based convolution and heuristic matching. *ACLWeb*, (P16-2022).

Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hananneh Hajishirzi. 2017. Bi-directional attention flow for machine comprehension. *arXiv*, (preprint arXiv:1611.01603).

Shuohang Wang, Mo Yu, Xiaoxiao Guo, Zhiguo Wang, Tim Klinger, Wei Zhang, Shiyu Chang, Gerald Tesauro, Bowen Zhou, , and Jing Jiang. 2017. R^3: Reinforced ranker-reader for open-domain question answering. *arXiv*, (preprint arXiv:1709.00023).