QA with Wiki: improving information retrieval and machine comprehension

Rohan Sampath
Department of Computer Science
Stanford University
rsampath@stanford.edu

Puyang Ma
Department of Statistics
Stanford University
pym86@stanford.edu

Abstract

The open-domain question answering task has recently been addressed using (a) structured data in knowledge bases, (b) unstructured data such as websites and online encyclopedias, and (c) a combination of both. An example of (b) is the DrQA system by Chen et al. [2017], where open-domain questions are answered through (a) finding Wikipedia articles relevant to the question (*document retrieval*), and then (b) predicting an answer span within the retrieved articles (*machine comprehension of text*). This paper proposes to extend the DrQA implementation for both document retrieval and machine comprehension. For the former, we experiment with incorporating PageRank when computing the "document score" to assess relevance of a document to a query. For the latter, we experiment with manual features and with implementing the BERT architecture based on Devlin et al. [2018]. Our experiment with PageRank adjusted retriever suggests it is a promising direction to work on for performance improvement of the retriever module. BERT dramatically improves performance for machine comprehension; however, there is some evidence to suggest that performance gains aren't as strong on answers with long spans.

1 Introduction

Open-domain question answering has traditionally been defined as finding answers in collections of unstructured documents (Chen et al. [2017]). Our motivation in tackling this problem lies in our interest in Natural Language tasks that have widespread application in society. The task of question answering is already ubiquitous - think of Alexa, Siri, and other virtual assistants. One intriguing aspect of these virtual assistants: to some questions, they reply with an answer (i.e., perform both doc retrieval and machine comprehension); in other cases, they merely return the top results from Google (i.e., use Google to perform information retrieval). This suggests a gap in performance today for open-domain QA, and indicates that truly useful virtual assistants will need to achieve better performance on the *sequence of* information retrieval and machine comprehension together - not just one, or both considered separately.

2 Related Work

The advent of structured knowledge bases such as Freebase (Bollacker et al. [2008]), has spawned systems to solve open-domain QA problem based solely on these knowledge bases. Other systems, such as IBM's sophisticated DeepQA for Jeopardy (Ferrucci et al. [2010]), train their QA models on both structured knowledge bases as well as unstructured data, such as websites.

For our implementation, however, we extract answers from purely unstructured data from a single source, Wikipedia. Wikipedia has been used as a corpus for open-domain QA extensively in literature;

a recent example of this is Ryu et al. [2014]. The authors use Wikipedia articles, but combine it with myriad other semi-structured data sources and other linguistic and semantic information such as definitions and article structure.

In this paper, adapt the work of Chen et al. [2017] and use Wikipedia for two reasons. First, the corpus is comprehensive enough (i.e., doesn't suffer from significant domain-specific knowledge gaps - particularly for everyday questions); however, it is also relatively homogeneous in structure and not prohibitively large. Second, the corpus is also the most frequently updated, which ensures up-to-date answers to questions.

The Chen et al. [2017] DrQA model has another salient feature - a pipeline that independently trains the information retrieval engine, called the "Retriever", and the machine comprehension engine, called the "Reader", and then runs them sequentially at test time. This also has strong basis in literature; the aforementioned DeepQA system uses a similar pipeline structure. Another noteworthy example of this is the open-source YodaQA model by Baudiš [2015], which is modelled after DeepQA, and also combines information from websites and databases, including Wikipedia.

3 Approach

Below, we describe our approaches to document retrieval (*Document Retriever*) and machine comprehension (*Document Reader*) as adapted from Chen et al. [2017].

3.1 Document Retriever

The original DocRetriever measures the similarity between a given query and articles by using dot product of TF-IDF weighted bag-of-word vectors. The vectors are computed after hashing bigrams to 2^{24} bins with unsigned murmur3 hash, where murmur3 is a non-cryptographic hash function that hashes bigram tokens to bins and similar tokens hashed to the same bin. Taking inspiration from popular search engines like Google, we thought to include PageRank scores as an additional relevance measure. The idea is that although the DocRetriever considers the similarity between a query and the text of an article, it does not account for links or relevance among articles. The retriever's performance may improve if we include this additional factor. To combine the two measures, we chose to use a weighted sum.

$$\mathbf{c_1} * \mathbf{p^T q_{doc}} + \mathbf{c_2} * \mathbf{log}(\mathbf{PageRank_{doc}} + \mathbf{1})$$

where $\mathbf{c_1}, \mathbf{c_2} \in R^+$ and $\mathbf{p}$ is the TF-IDF vector representation of a query, $\mathbf{q_{doc}}$ the TF-IDF vector of a document. $\mathbf{log}(\mathbf{PageRank_{doc}} + \mathbf{1})$ to get small PageRank values (on the scale of 1e-6) comparable to the dot product. The PageRank score of a document is irrelevant with the query, but measures the importance of an article in the network. Therefore, it accounts for the links between articles. We utilized pre-computed PageRank scores of Wikipedia articles from a GitHub repository Thalhammer.

3.2 Document Reader

DrQA approach We largely adopt the approach of the DrQA system by Chen et al. [2017]. The details can be found in that paper, but we suummarize the appoachg below:

Given a question q consisting of k tokens $q_1, ..., q_k$ and a document or a small set of documents of n paragraphs where a single paragraph p consists of m tokens $p_1, ..., p_m$, we use an LSTM model that we apply to each paragraph in turn and then finally aggregate the predicted answers. The three layers that make up the LSTM are as follows:

I. **Paragraph encoding:**
- 300 dimensional GloVe embeddings
-"f_{exact}: a *manually added* binary feature that checked whether the paragraph word can be matched to a question word
- Token features: part-of-speech tagging, named entity recognition and term frequency
- Aligned question embedding ($f_{aligned}$): captures similarity between paragraph words and each question word through question-to-context and context-to-question attention

II. **Question encoding:** applies an RNN on top of the word embeddings of the words in the question.
III. **Prediction layer:** two classifiers that independently predict the two ends of the span (with the span being at most 15 words).

Feature engineering We add two manual features: (a) Context word is the antonym of a question word, based on WordNet, and (b) Context word is the synonym of a question word, based on WordNet.

In the ablative analysis performed in Chen et al. [2017], removing the manually added f_{exact} as well as $f_{aligned}$ dramatically reduced performance. However, removing either one reduced the F1 score by <2 pointsm indicating that these features were complementary. What we therefore wanted to test is if similar complementarity exists between "synonym" and "antonym" features and $f_{aligned}$, or whether there was additional information conveyed to the model by including these manual features. Our initial hypothesis was that, since $f_{aligned}$ measures soft alignments between context words and question words through attention, it would *implicitly* measure synonymy and antonymy; however, an *explicit* measure of synonymy and antonymy is likely to provide some modest improvements over this implicit measure. The idea for these additional manual features was adapted from Yerukola and Kamath [2018], but we have written our own code to implement these features.

BERT model Next, we implement the BERT model, by Devlin et al. [2018] as our Reader. Unlike unidirectional models (including Vanilla RNNs and LSTMs) BERT - which stands for Bidirectional Encoder Representations from Transformers - pre-trains deep bidirectional representations by jointly conditioning on both the left and right context in all layers. The BERT implementation of a multi-layer bidirectional Transformer is based on the original Transformer implementation specified in Vaswani et al. [2017].

BERT pre-trains a model by optimizing for two language model tasks: (i) "masked" word prediction, and (ii) next sentence prediction. Details of these pre-training tasks can be found in section 3.3 of Devlin et al. [2018]. We use the BERT-base-uncased model (which contains 110M parameters), as our pre-trained model, which we download from the huggingface repository.[1].

We then fine-tune this model based on our specific, supervised downstream task - question answering for SQuAD 1.1.

4 Experiments

4.1 Document Retriever

Data For the DocRetriever, we updated the 2016-12-21 English Wikipedia dump used in the DrQA paper to the 2019-03-02 English Wikipedia dump. For a fair comparison between the original DocRetriever's performance and our new retriever model's performance, we used WikiExtractor and wrote a Java script to extract 100,000 Wikipedia articles as the source of answers for a given query. For evaluation, we used the same datasets as in the DrQA paper by Chen et al. [2017], namely SQuAD 1.1 dev dataset, WebQuestions test set, and CuratedTrec test set to test the two retriever models' performance.

Evaluation Currently we keep using the evaluation metric used in the DrQA paper, namely the percentage of questions for which the answer segment appears in one of the top 5 pages returned by the retriever.

Experimental details Both the DocRetriever model (DocRetriever+bigram) and our PageRank adapted retriever (Retriever+pagerank) are evaluated on the same dataset: SQuAD 1.1 dev. For the PageRank adapted retriever, we experimented with several combinations of c_1 and c_2 (section 3.1), and eventually found that the best weights would be the following: $c_1 = 0.5$ and $c_2 = 0.5$.

Results The table shows evaluation results of the two retrievers. Due to smaller size of the Wikipedia corpus (about 2% of that used in the DrQA paper), the DocRetriever's performance is lower than that reported in the paper. Our model with the PageRank addition shows a small performance improvement.

[1]https://github.com/huggingface/pytorch-pretrained-BERT

	SQuAD 1.1	WebQuestions	CuratedTrec
DocRetriever+bigram	69.7	66.9	30.6
Retriever+PageRank	70.1	67.5	30.6

Table 1: Results of the two retriever models on multiple datasets. % of questions for which the answer segment appears in one of the top 5 pages returned by the model.

4.2 Document Reader

Data We use SQuAD 1.1 (Rajpurkar et al. [2016]) for training and testing our Document Reader. We did not use SQuAD 2.0 (Rajpurkar et al. [2018]) because SQuAD 2.0 purposefully contains a number of (paragraph, question) pairs for which the answer to the question does not lie in the paragraph, and penalizes a model that produces an answer to such questions. However, since we are developing a combined Retriever-Reader system, we *do* want all questions to be answered, by retrieving the correct Wikipedia articles and then successfully finding the answer within the retrieved documents. Therefore, we do not want to train a Reader on SQuAD 2.0 since that would frequently predict "no answer".

The SQuAD 1.1 dataset consists of a number of articles; each article contains a number of paragraphs, and each paragraph contains a number of question-answer pairs. As mentioned above, there are no unanswered questions. Five Amazon Mechanical Turk annotators annotate the answers to each question - these five answers are known as "ground truths" (it is often the case that multiple annotators annotate the same answer). Summary statistics are found in Table 2 below:

	SQuAD 1.1	
	Train	Dev
Number of articles	442	48
Number of paragraphs	18,896	2,067
Number of questions/answers	87,599	10,570

Table 2: Summary Statistics of SQuAD 1.1 train and dev datasets

Evaluation We report the two standard evaluation metrics on the dev set - exact match and F1 score. For each question, the score is the maximum across all ground truths (i.e., if the answer predicted by the model exactly matches any one ground truth, it is counted as an exact match).

Experimental details We compare the performance of three models: (a) replicating the DrQA baseline, (b) DrQA + feature engineering with synonyms and antonyms as features, (c) BERT model fine-tuned on SQuAD 1.1 For (a) and (b), we largely follow the implementation approach described in Chen et al. [2017]. The details can be found in section 5.2 of the paper, but the salient details are as follows: (i) A 3-layer bidirectional LSTMs with h = 128 hidden units for both paragraph and question encoding, (ii) Stanford CoreNLP toolkit (Manning et al. [2014]) for tokenization and generating lemma, part-of-speech, and named entity tags, (iii) mini-batch size of 32, (iv) Adamax for optimization, and (v) dropout of 0.3.

For (c), we used the BERT-base-uncased model as the pre-trained model, and fine-tuned it on SQuAD 1.1. We tuned four hyperparameters: (a) Batch size (b) Max sequence length (maximum length of input sequence, in terms of number of tokens) (c) Gradient accumulation steps (number of updates steps to accumulate before performing a backward/update pass), and (d) Learning rate. After testing various combinations of hyperparameters, we found that the following combination yielded the best results: (a) Batch size of 6, (b) Max sequence length of 384, (c) Gradient accumulation step of 1, (d) Learning rate of 2e-5. We ran our experiments on Azure's NV6 virtual machine, which contains a single NVIDIA Tesla M60 GPU with a memory of 8 GB.

Results The results of our experiments are presented in Table 3.

Model	SQuAD 1.1	
	EM	F1
DrQA DocReader	69.3	78.6
DrQA DocReader + synonym + antonym features	69.5	78.9
BERT Reader (fine-tuned)	81.8	88.6

Table 3: Results of the three Reader models, on SQuAD 1.1's dev dataset.

First, the feature engineering that we performed had a limited effect. Our hypothesis: the DrQA model by Chen et al. [2017] already includes the $f_{aligned}$ feature which uses attention to measure *soft alignments* between context words and question words. This is in some ways a proxy for synonymy and antonymy, and therefore the addition of synonym and antonym features does not provide much additional information to the model. Second, as indicated by Devlin et al. [2018], the BERT Reader, fine-tuned on SQuAD 1.1, produced dramatically better performance; we achieved a 12 and 10 point improvement on EM and F1 scores respectively.

4.3 Retriever-Reader Pipeline

Evaluation We run two combined Reader-Retriever pipelines: (a) Baseline: DrQA Retriever + DrQA Reader, and (b) DrQA and PageRank retriever + DrQA Reader with synonym, antonym features. For each pipeline, we test the following: (a) top 2 docs returned by Retriever, and (b) top 5 docs returned by Retriever. The Reader scans all the documents returned by the Retriever and makes its best prediction. Our evaluation metric is "best prediction exact match accuracy" - whether the best prediction made by the Reader exactly matches any of the ground truths in the SQuAD 1.1 dev set.

Experimental Details The experimental parameters that we found worked best were as follows: (a) Paragraph batch size of 128, (b) Prediction batch size of 128. Evaluation takes 6-8 hours to complete for each run of the pipeline, depending on the number of documents being returned by the Retriever.

Results The results for the combined Retriever-Reader pipeline are reported below:

Model	SQuAD 1.1	
	Top-2 docs	Top-5 docs
DrQA[2]	22.3	23.2
DrQA + PageRank Retriever + Reader with synonym, antonym features	22.5	23.4

Table 4: Best prediction exact match %, for the two pipelines on SQuAD 1.1's dev set

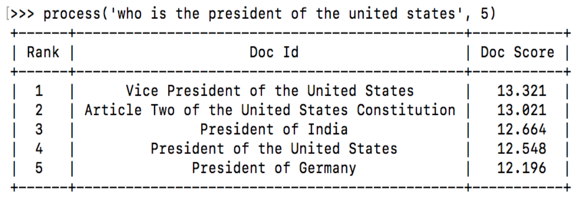
5 Analysis

5.1 Document Retriever

The inclusion of PageRank with optimal weights showed some improvements on the retriever's performance in SQuAD v1.1 dev set and WebQuestions test set as shown in section 4.1.4. In addition to the improved number of returned articles containing the answer span, we observed qualitatively, the PageRank modified retriever tends to fetch articles more relevant to a given question than that fetched by the DocRetriever, as shown by the following sample query results(Figure 1). To the query "Who is the president of the United States", the PageRank modified retriever returns the article "President of the United States" as the top choice versus being the 4th choice by the DocRetriever. Moreover, a direct answer to the query "Ronald Reagan" is among the top 5 returned by the modified retriever. In addition to short answer questions, the PageRank modified retriever also performed well for more complex questions like "Why does coffee taste bitter?". The modified retriever returned

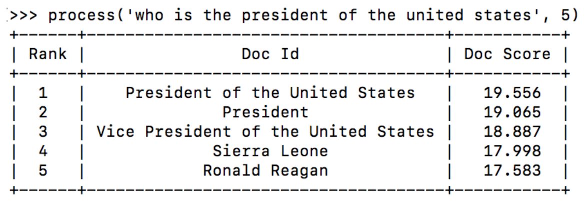
[2]As mentioned in section 4.1, we run our DocRetriever on 100K Wikipedia articles rather than the 5M that DrQA uses, due to time constraints, where these 100K are chosen randomly from the 5M. Therefore, this baseline score does not match that of DrQA Pipeline reported in Chen et al. [2017], which achieved a best prediction exact match accuracy of 27.1.

"Tea" as one of the top 5 articles, which is more relevant with the question than "Casein" returned by the DocRetriever. In conclusion, the precision of retrieved articles to a given query is not captured by the percentage of top 5 documents containing the answer span. Some articles may contain the answer by chance while others not only contain the answer but directly address the question. The modified retriever is able to capture the latter type of documents.



```
>>> process('who is the president of the united states', 5)
+------+------------------------------------------+-----------+
| Rank |                  Doc Id                  | Doc Score |
+------+------------------------------------------+-----------+
|  1   |      Vice President of the United States |   13.321  |
|  2   | Article Two of the United States Constitution |   13.021  |
|  3   |            President of India            |   12.664  |
|  4   |       President of the United States     |   12.548  |
|  5   |           President of Germany           |   12.196  |
+------+------------------------------------------+-----------+
```

(a) DocRetriever



```
>>> process('who is the president of the united states', 5)
+------+-------------------------------------+-----------+
| Rank |                Doc Id               | Doc Score |
+------+-------------------------------------+-----------+
|  1   |    President of the United States   |   19.556  |
|  2   |              President              |   19.065  |
|  3   | Vice President of the United States |   18.887  |
|  4   |             Sierra Leone            |   17.998  |
|  5   |            Ronald Reagan            |   17.583  |
+------+-------------------------------------+-----------+
```

(b) Modified Retriever

Figure 1: Top 5 articles retrieved to sample queries by DocRetriever and the modified retriever

5.2 Document Reader

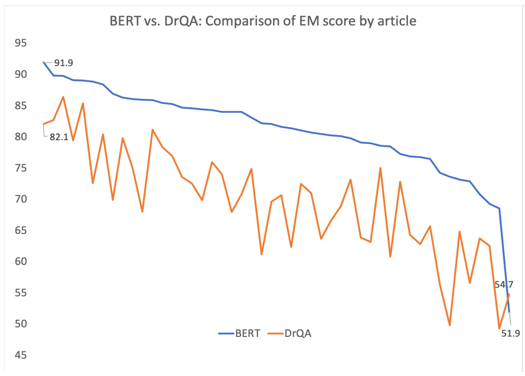
Shown below is a contingency table that compares the exact match accuracy of BERT vs DrQA:

		DrQA DocReader	
		Correct	Incorrect
BERT Reader	Correct	$6,790$ (64.2%)	$1,851$ (17.5%)
	Incorrect	542 (5.1%)	$1,387$ (13.1%)

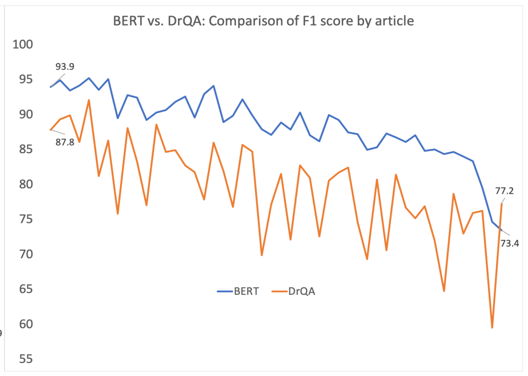
Table 5: Contingency table of number and percent questions answered correctly (i.e., exact match) by each Reader. Results are reported on SQuAD 1.1's dev set, which has 10,570 questions.

The bottom left quadrant captures an interesting insight: for 5.1% of questions, DrQA predicts the answer correctly while BERT predicts it incorrectly. In other words, a hypothetical "best of DrQA and BERT" model would achieve an EM score of 86.9 - very close to human-level performance.

We therefore investigate the performance of BERT vs. DrQA at an article level to see whether there are certain articles that DrQA is better at, and what the characteristics of those articles might be.



(a) EM score comparison



(b) F1 score comparison

Figure 2: Article-by-article comparison of EM and F1 scores produced by BERT vs DrQA for the 48 articles in SQuAD 1.1. Articles are ordered from highest to lowest BERT EM score for both graphs.

In Figure 2, we find that BERT does better on both the EM and F1 metrics for all articles except one, titled "Packet Switching". A visual inspection of "Packet Switching" and its questions suggests that it is an outlier in terms of answer length - the annotated "ground truth" answers to questions in "Packet

Switching" have an average length of 7.4 tokens, as opposed to 2.9 for the SQuAD 1.1 dev set as a whole. Therefore, we next investigate if there is a relationship between the average length of the answers to an article, and the F1 score improvement that BERT achieves on the article.

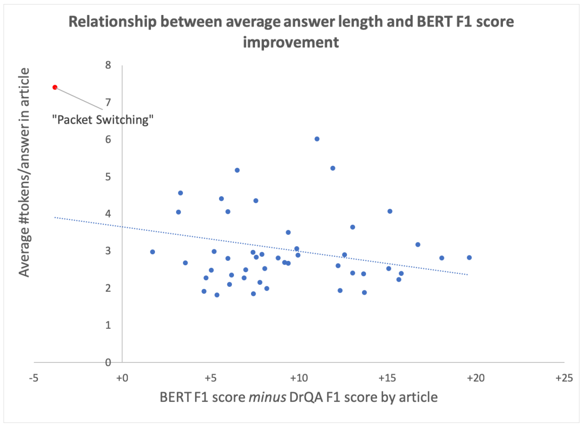


Figure 3: Average # of tokens/answer vs. difference between BERT F1 and DrQA F1, by article.

Unfortunately, the plot in Figure 3 inconclusive - primarily because most articles are clustered around the 2-4 tokens/answer range, and very few articles that are >5 tokens/answer. However, we believe this to be a worthwhile avenue for further exploration using other datasets with longer answer lengths. One hypothesis could be that, when answers in the training set are long, attention in the multi-headed attention layers of BERT become too "diluted" or "diffused". If it is true that the performance improvements from BERT reduce as the length of answer spans increase, that has significant implications for general purpose open-domain question answering.

5.3 Retriever-Reader Pipeline: the "fit" score

One of the most interesting insights from Chen et al. [2017] was the discrepancy between between the performance of the individual Retriever and individual Reader vs. the overall Reader-Retriever pipeline. Specifically, 77.8% of questions had the answer appear in at least one of the top 5 pages returned by DrQA DocRetriever; let this metric be called $EM_{retriever}$. The EM score on DocReader alone, when trained on SQuAD 1.1 (henceforth EM_{reader}) was 69.5. The ideal EM score on the entire pipeline (henceforth $EM_{pipeline}$) would be expected to be $EM_{retriever} \times EM_{reader} = 77.8 \times 69.5 = 54.1$. However, in reality, $EM_{pipeline}$ (without distant supervision learning) was 24.5.

We therefore introduce the concept of a Retriever-Reader "fit score", γ, which is defined as follows:

$$\gamma = EM_{pipeline}/(EM_{retriever} * EM_{reader})$$

In other words, rearranging the above equation, we have that the $EM_{pipeline}$ is the product of three factors: (a) $EM_{retriever}$: the performance of the retriever individually (b) EM_{reader}: the performance of the reader individually, and (c) γ: How good the "fit" between the two is.

We report results for the two models specified in Table 4, assuming that the Retriever retrieves the top 5 articles; as previously, results are reported for 100K Wikipedia articles.

Model	SQuAD 1.1			
	$EM_{retriever}$	EM_{reader}	γ	$EM_{pipeline}$
DrQA	69.7	69.3	48.0	23.2
DrQA + PageRank Retriever + Reader with synonym, antonym features	70.1	69.5	48.0	23.4

Table 6: Comparison of metrics across the two pipelines on SQuAD 1.1's dev set

Since the two pipelines above do not vary significantly in performance, the γ values are also effectively the same. It would be instructive to see the impact of using BERT Reader on γ - for example, while the individual BERT reader performs very well, the combined Reader-Retriever pipeline may not see the same increase in performance.

Qualitatively, γ is less than 100% for the following reasons; one potential research topic would be to decompose the effects of each of these on γ:
a) Flawed $EM_{retriever}$ score: the "EM Retriever" score reflects whether any of the documents returned contain the correct answer span. Consider this example: the question is "what is the answer to everything in the universe", and the retriever returns an article which happens to have the number 42 anywhere in the article; the $EM_{retriever}$ score would count this as a match
b) Different corpora: DocReader is trained on SQuAD; this trained model is then used to make a prediction for the most likely span among returned Wikipedia articles. Clearly, since there are substantial differences in these corpora, DocReader will not perform as well on Wikipedia articles
c) True Reader-Retriever fit: given that the Retriever and Reader are trained independently of each other, it is not the case that their joint performance is being optimized for

6 Conclusions and Future Work

Conclusions In summary: (a) including PageRank has a small effect on improving the performance of DocRetriever on SQuAD and WikiMovies, but not on CuratedTREC; (b) using BERT fine-tuned on SQuAD 1.1 dramatically improves performance over the DrQA DocReader; (c) there is some evidence to suggest that BERT's performance improvements aren't as substantial with questions having longer answers; this needs to be tested on datasets other than SQuAD; and (d) we introduce the concept of a "fit" score γ that bridges the gap between the individual evaluation of the Retriever and Reader, and the evaluation of the Retriever-Reader pipeline as a whole.

Future Work: Overall The immediate next step would be to test the performance of the Retriever-Reader pipeline with a BERT Reader.[3] We would also like to research how the fit score γ can be decomposed into its constituent parts - evaluation metric flaws, dataset differences, and true retriever-reader model fit.

Future Work: Document Retriever The immediate next step would be to extend this analysis to all 5M Wikipedia articles used in the original DrQA paper by Chen et al. [2017]. This would require us to optimize our PageRank algorithm to compute scores for 5M Wikipedia articles rather than 100,000, which would require higher-performance compute resources. Second, we think that topic-specific PageRank, which includes considerations of a given query's topic, can further improve the Retriever's performance.

Future Work: Document Reader The natural next step is to experiment with improving the BERT architecture. Recent work on improving BERT for question answering, such as that by Wu et al. [2019], could serve as initial guidance. We would use the "fit" metric γ we introduce in this paper to evaluate any modifications to the BERT architecture, so as to ensure that we are not just optimizing for individual Reader performance, but instead for the entire pipeline.

7 Additional Information

Our advisor for this project is Professor Christopher Manning.

References

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

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[3]Unfortunately, the complexity and time required to convert the BERT model implementation into a format that was accepted by the DrQA pipeline proved to be prohibitive for this project.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A Kalyanpur, Adam Lally, J William Murdock, Eric Nyberg, John Prager, et al. Building Watson: An overview of the DeepQA project. *AI magazine*, 31(3):59–79, 2010.

Pum-Mo Ryu, Myung-Gil Jang, and Hyun-Ki Kim. Open domain question answering using wikipedia-based knowledge model. *Information Processing & Management*, 50(5):683–692, 2014.

Petr Baudiš. Yodaqa: a modular question answering system pipeline. In *POSTER 2015-19th International Student Conference on Electrical Engineering*, pages 1156–1165, 2015.

Andreas Thalhammer. Pagerank calculation for millions of wikipedia articles. `https://github.com/athalhammer/danker`.

Akhila Yerukola and Amita Kamath. Adversarial SQuAD. *CS 224N Winter 2018 Final Project*, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

Pranav Rajpurkar, Robin Jia, and Percy Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. *arXiv preprint arXiv:1806.03822*, 2018.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60, 2014.

Felix Wu, Boyi Li, Lequn Wang, Ni Lao, John Blitzer, and Kilian Q Weinberger. Fastfusionnet: New state-of-the-art for dawnbench squad. *arXiv preprint arXiv:1902.11291*, 2019.