An Exploration and Evaluation of DrQA Variant Implementations

Avinash Nayak
Princeton University
anayak@princeton.edu

Kai Lu
Princeton University
kail@princeton.edu

Navaneethan Vaikunthan
Princeton University
nv5@princeton.edu

Abstract

DrQA is a recently developed system for question answering from Wikipedia articles, and its aim is to build QA models for information sources that are primarily read and analyzed by humans, rather than by computers (i.e. Knowledge Bases). It is composed of two components: one that uses information retrieval to find relevant documents, and one that uses multi-layer recurrent neural networks for machine comprehension in finding answer spans in text. Our project aimed to develop and test the efficacy adjustments to the DrQA method, in particular to employ a novel approximate nearest neighbor (ANN) based approach with locality sensitive hashing to retrieve relevant documents in reasonable runtime, using embeddings that were based on tokenized paragraphs rather than articles, suggested by the summary paper of Mikhail Khodak in section 4.1 [1]. For our results, we used three different retrievers: one with the original DrQA implementation and flexible threshold cutoff (baselines), an ANN retriever with size 50 word embedding vectors, and an ANN retriever with size 300 word embeddings. While the baseline retriever held the best performance in accuracy, our other approaches represent an effort to bring novel hashing and algorithmic techniques to document retrieval, past the standard TF-IDF approach. Furthermore, our results spark impetus for future exploration in evaluating the efficacy of current state-of-the-art word embedding models in modeling question-answer data, and other more complex data relationships than correlating single words.

1 Introduction

Today, there exists a sizable imbalance between the amount of information that people crave at their fingertips and the memory capacity that people possess. Hence, the research area of open-domain question answering (QA) has become very relevant. In contrast with QA from knowledge bases (KBs), such as Freebase or DB-pedia, which are easier for computer processing but too sparse in their domain breadth, open-domain QA from websites like Wikipedia is designed to answer questions that everyday laypeople have, rather than solely domain experts. This characteristic allows open-domain QA systems to have the ability to serve a wide user base, and is evident in multiple virtual assistants such as Microsoft's Cortana and Apple's Siri.

This paper is based on DrQA [2], a system for question answering from Wikipedia articles, which is composed of two separate and autonomous components. The first is a Document Retriever, which when given a question as input, uses bigram hashing and TF-IDF to output a collection of articles relevant to the input question. Following this module, the second component is a Document Reader, which uses a multi-layer recurrent neural network (RNN) to be able to locate spans (text segments) of answers within the documents that have been returned from the Document Reader. Within this system, the modules for information retrieval and machine comprehension are distinct.

In this work, we seek to research, implement, and analyze potential enhancements to DrQA, particular to those described in the summary paper of DrQA by Mikhail Khodak [1]. Specifically, we implement changes particular to Section 4.1 of [1], in which we test the efficacy of retrieving paragraphs for embedding representations, rather than articles, and we use a novel approximation method for our retrieval in order to achieve reasonable runtime. We also test our retrieval accuracy with different vector sizes for our word embeddings, and change our cutoff from being arbitrary to being based on paragraph cosine similarity. Our methodology and evaluation of these aforementioned approaches can provide insight into novel future techniques that can be used in document retrieval and understanding for open-domain QA, such as evaluation of the efficacy of state-of-the-art generalized word embedding models in modeling QA data.

2 Related and Relevant Work

2.1 DrQA

While DrQA currently remains to be the state-of-the-art system for open-domain QA, there have been many prior models and research efforts that have paved the path for the fruition of DrQA. While the initial development of KBs, including resources such as WebQuestions and SimpleQuestions which were based on Freebase KB, served as a baseline for QA input data, it was realized that KBs have certain limitations with respect to the limited breadth of their schemas. Researchers subsequently surmised that in order for machines to best replicate humans in their answering of questions, techniques should be reset to answering questions from raw text, as humans would read.

Within the DrQA paper [2], the researchers utilize the recent advancements in memory-based deep learning architectures to build an RNN model for question answering based on retrieved documents. Figure 1 from the DrQA paper provides an overview of the architecture.

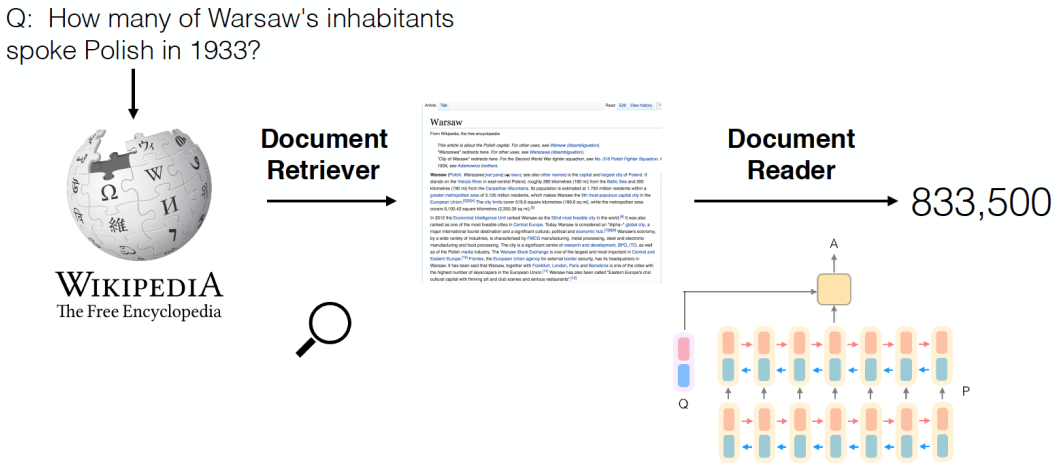

Figure 1: Derived from [2]. An overview of the DrQA system.

A brief description of each their modules is provided in subsections 2.1.1 and 2.1.2.

2.1.1 Document Retriever

The document retriever module is used to narrow the search space of articles. In addition to using TF-IDF bag-of-word vectors, Danqi's paper[2] investigates local word order using n-gram counts, with bigrams providing the highest performance. In their implementation, the Document Retriever returns five Wikipedia articles after being given a question, but our approach does not return an arbitrary threshold, but rather one based on cosine similarity of paragraphs. Our procedure will be detailed further in section 4 of this paper.

2.1.2 Document Reader

While the Document Retriever module did not use any machine learning techniques, the Document Reader is based on state-of-the-art neural network models, which have achieved increasing success in machine comprehension tasks. A detailed description of the architecture of the RNN model that is used is provided within [2].

2.2 Approximate Nearest Neighbors

For our research, we deal with large collections of high dimensional embedding vectors that represent paragraphs, and nearest neighbor algorithms become too computationally expensive for the upper levels of dimensionality that we use. Hence, we invoke an approximation solution, namely Annoy (Approximate Nearest Neighbors Oh Yeah) [3], which is a C++ library with certain Python bindings, representing a variant for the nearest neighbor search. This library, which we employ in our implementation, uses a family of techniques known as Locality Sensitive Hashing (LSH), which is a dimensionality reduction technique for data of high dimensions. More specifically, LSH allows for defining a hash function which tosses word vectors into bins, where each bin is allocated with a probability measure that is proportionate to the cosine similarity between vectors placed in the bin.

3 Data

The novelty of this paper exists in our implementation adjustments, our application of approximate nearest neighbor techniques to document retrieval, and our analysis of word embeddings over paragraphs rather than documents. As a result, we decided to treat our original data as a control variable, and we used the same datasets as the original DrQA paper [2].

3.1 SQuAD

The Stanford Question Answering Dataset (SQuAD) is a newly developed dataset from 2016 that is based on questions accumulated by crowdworkers on sets of Wikipedia articles. Structurally, the answer to every question is a segment of text (span) from the reading passage input, and the dataset consists of over 100,000 question-answer pairs derived from more than 500 articles. This expanse of information makes SQuAD much larger and more detailed than previously compiled reading comprehension datasets, including Microsoft's MCTest dataset, consisting of 500 stories and solely 2000 questions. Figure 2 is sample data taken from Rajpurkar's paper on SQuAD [4].

In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail… Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 2: Adapted from [4]. A sample of question-answer pairs for a test passage, with each answer represented by a span, or text segment.

3.2 Other Data

In addition to SQuAD, we used other datasets that were used in [2], namely CuratedTREC, WebQuestions, and WikiMovies, all of which are summarized in detail in [2].

Dataset	Example	Article / Paragraph
CuratedTREC	**Q**: What U.S. state's motto is "Live free or Die"? **A**: New Hampshire	**Article**: Live Free or Die **Paragraph**: "Live Free or Die" is the official motto of the U.S. state of New Hampshire, adopted by the state in 1945. It is possibly the best-known of all state mottos, partly because it conveys an assertive independence historically found in American political philosophy and partly because of its contrast to the milder sentiments found in other state mottos.
WebQuestions	**Q**: What part of the atom did Chadwick discover?† **A**: neutron	**Article**: Atom **Paragraph**: ... The atomic mass of these isotopes varied by integer amounts, called the whole number rule. The explanation for these different isotopes awaited the discovery of the neutron, an uncharged particle with a mass similar to the proton, by the physicist James Chadwick in 1932. ...
WikiMovies	**Q**: Who wrote the film Gigli? **A**: Martin Brest	**Article**: Gigli **Paragraph**: Gigli is a 2003 American romantic comedy film written and directed by Martin Brest and starring Ben Affleck, Jennifer Lopez, Justin Bartha, Al Pacino, Christopher Walken, and Lainie Kazan.

Figure 3: Adapted from [4]. A sample of question-answer pairs for a test passage, with each answer represented by a span, or text segment.

4 Methods and Implementation

4.1 Database Transformation

Since the original implementation uses bigram hashing and tf-idf to find the closest documents, the retriever only needs to pull raw text, since bigram counts can easily be calculated at run time. However, when using an ANN algorithm like LSH to find the closest documents, all documents need to already have been indexed prior to runtime. Indexing of documents must occur before runtime, since the random projection algorithm for LSH requires the construction of several trees in order to support a search algorithm. Thus, the first step in our implementation requires that we convert the raw text of the document to a fixed vector representation.

To align with the Document Reader portion of DrQA, it makes the most sense to convert documents to a GloVe embedding based representation. In order to perform this, we used pre-trained embeddings from the Common Crawl as the basis for the representation. Due to technical difficulties explained later, both size 300 and size 50 GloVe embeddings were used for the following transformation. These embedddings are associated with words, and as such a decision had to be made with regards to how to aggregate to the document level. In order to be agnostic to the size of the document itself, this implementation takes the mean of the word embeddings contained in a document to represent it. However, applying the mean over an entire document will likely introduce too much noise to the representation. To get around this limitation, the corpus is split by paragraph before applying this transformation. As an additional noise reduction method, a list of ~ 100 common English words were used as stopwords that were filtered out of the raw text before conversion. With these implementation details decided upon, we proceeded to convert the raw text database into a database of GloVe-based paragraph vectors.

4.2 Pipeline

After the new database was constructed, the pipeline itself also needed to be transformed such that paragraphs could now be retrieved through our new approach. As mentioned before, ANN algorithms such as the one employed by the annoy library require data structures to be built prior to query time. In order to do this, the paragraph vectors from our database are utilized to create k separate hash trees. As k grows, the accuracy of the ANN method grows, but at the cost of increased memory consumption. After tuning this hyperparameter, the choice was made to build 10 hash trees.

With the trees built, the rest of the pipeline works in a similar manner to the original implementation, but with a few important changes. First, the query must itself be converted to its new dense embedding format by taking the mean of the GloVe embeddings associated with the query. The query is filtered by the same list of stopwords before this conversion in order to maintain similarity to the

paragraph database. After the query is properly converted, we then us the annoy library to find the closest paragraph vectors to our query. The annoy library's implementation of ANN allows us to pull a given number of paragraphs so long as the cosine similarity is over a pre-defined cutoff. Thus, our pipeline allows the retriever to pull a non-arbitrary amount of paragraphs from our database, while maintaining a reasonable run time.

4.3 Technical Challenges

Since the database is constructed from the entirety of Wikipedia's text, there are many technical issues that arose from the size of the data set itself. The first issue is in the construction of our paragraph vector database, which needs to store 37 million paragraph vectors in total. To associate a size n vector with a given paragraph, a naive implementation would require $20n$ bytes of space. However, by converting the vector to a string and applying standard string compression, the space used by a size n vector can be taken down to $\sim 5n$ bytes. Even with compression, the default size 300 vectors would take up on the order of 55.5 GB of space which is doable for most household and enterprise machines, but still an issue.

The bigger technical issue stems from how annoy is able to search for the closest documents to a given query string. In order to search for close documents, the library must have access to all asscoiated hash trees in memory rather than on disk. The memory cost is approximately proportionate to knN where k refers to the number of hash trees, n refers to the size of the vector, and N refers to the amount of vectors in the database. For size 300 vectors, this means that the trees take more than 50 GB of memory, which is unreasonable to expect from a standard household or enterprise machine.

In order to circumvent this issue, there were two parallel soultions, each with its own set of advantages and drawbacks. The first of these solutions was simply to use size 50 vectors, which would result in the trees being on the order of 10 GB, which is much more reasonable for most computers. However, size 50 vectors have less information than its larger counterparts, meaning that this approach may cause accuracy to suffer. Another way in which to get rid of this problem is to put split the database into eight sections, each with their own associated annoy hash tree structures. In order to search, this requires each tree to be searched seperately and their results collated. While this method allows us to use the default size 300 vectors, looping through each tree structure requires more run time. Moreover, this process cannot be parallelized as it causes a segmentation fault, meaning that the process must operate on only a single thread. All in all, this results in a much longer run time, which is not ideal in a QA system.

Tf-idf Cutoff	
150	69.22% (373.39 s)
200	77.12% (372.90 s)

Table 1: Results for 'Baseline' with tf-idf cutoff

5 Results

Results will be measured by two measures: accuracy and testing time. The accuracy measure is calculated by counting the number of times the correct answer to a given query is contained within the retrieved documents, and dividing this value by the total queries in the test set. Testing time refers to the length of time in seconds that it takes to evaluate the accuracy of our retriever on the entire SQuAD dataset of queries and answers, which acts as a proxy for understanding the query time for each retriever.

Four different types of retrievers are tested: a baseline retriever that uses the original implementation but with a flexible cutoff, our ANN-based retriever with size 50 vectors, our ANN-based retriever with size 300 vectors, and a retriever that uses tfidf to reduce the search space and then applies an ANN-based method for the ultimate choice. To further explain our 'baseline', this retriever is simply using the original database and pipeline, but instead of returning the top five documents it returns all documents over a given tf-idf score. There is one hyperparameter that is tuned in our

'baseline' retriever which is the value of our tf-idf cutoff. For both ANN-based retrievers, there are two hyperparameters which are cosine similarity cutoff and retrieval size. For our last retriever method, we use a multi-step method, in which the first step is to pull the documents or paragraphs with the n highest tf-idf score. After significantly reducing the search space in this manner, the last step is to use ANN methods to find all documents or paragraphs that are over a certain cosine similarity cutoff. As such, the hyperparameters that must be tuned are simply the number of documents to first retrieve using tf-idf and the cutoff to be used for the final Retrieval of documents.

Retrieval Size/Cutoff	.15	.20
200	29.87% (275.0 s)	29.84% (201.8 s)
1000	43.18% (942.3 s)	43.16% (799.2 s)

Table 2: Results for size 50 vectors

Starting first with our baseline, the accuracy for a tf-idf based retriever with a score cutoff of 200 is 69.22% with a testing time of 373.3 seconds. With a lower cutoff of 150, the accuracy jumps to a respectable 77.12%, which is actually nearly exactly the performance of the original implementation with had an accuracy of 77.8%. Next we examine the ANN-based retriever with size 50 vectors, which in general had inferior performance to its 'baseline' component. As seen in Table 2, at a Retrieval size of 200 the accuracy was a dismal $\sim$ 30% regardless of the cutoff. A Retrieval size of 1000 paragraphs fared better reaching around 43% accuracy for both cutoff points but at the cost of an increase in nearly 600 seconds in run time. Lastly, the ANN-based Retrieval models with size 300 vectors, which we hypothesized to be more effective than the previous implementations, ended up being even less effective than its size 50 counterpart. For a Retrieval size of 800 (100 for each tree looped through) and a cutoff of .20, the accuracy was 21.5% with a run time of around 36,000 seconds. A Retrieval size of 1600 (200 for each tree looped through) fared little better with an accuracy of 26.24% and a testing time of a almost 45,100 seconds. When reducing the cutoff to .15 and using a Retrieval size of 800 (100/tree), the results do not change much giving an accuracy of 21.4% with a run time of around 38,000 seconds. For our last retriever method, which is a multi-step process, a smaller test size of only $\sim$ 1000 of the total 100,000 must be used due to memory issues. This method was done in two manners, one that pulls paragraphs and another that pulls documents. Of the retrievers that first pull 20 paragraphs, the accuracy is around 37% for a cutoff of .20 and around 52% for a cutoff of .15. When increasing the amount of paragraphs first pulled to 30 and having the cutoff at .20, the accuracy does not increase much, settling at around 41% accuracy. Lastly, we tried retrievers that worked by pulling whole documents first instead of paragraphs. By setting the cutoff at .1 and pulling 20 full documents in the first step, the accuracy jumps to around 67.4%. Increasing the number of documents first pulled to 30, we get our last result of an accuracy of around 72%.

Retrieval Size/Cutoff	.15	.20
100 / tree (8 trees)	21.47% (8484.4 s)	21.47% (36859.6 s)
200 / tree (8 trees)		26.24% (45158.1 s)

Table 3: Results for size 300 vectors

First-Step Retrieval Size/Cutoff	.15	.20
20	54.10% (1072.3 s)*	37.64% (10323.5 s)
30		40.60% (1468.5 s)*

Table 4: Results for two-step retriever pulling by paragraphs

* The results in tables 4 and 5 that have an asterisk are using a subset 1000 queries of the SQuAD testing set. The entire set consists of 10,000 queries in total.

First-Step Retrieval Size/Cutoff	.1
20	67.41% (728.7 s)*
30	71.36% (1032.2 s)*

Table 5: Results for two-step retriever pulling by document

6 Discussion and Conclusion

Overall, our project started by implementing the three changes suggested in Mikhail Khodak's paper in section 4.1: changing the arbitrary 5 document cutoff to a non-fixed metric based cutoff, returning results at the paragraph level instead of the document level, and using word vectors as features. The results of implementing all three changes were very disappointing; not only did the new retriever require substantially more space and time (which was expected), overall the accuracy decreased dramatically compared to the original DrQA retriever. As a result, we shifted directions slightly to determine the most likely sources of inaccuracy. Our first implementation had 4 key differences compared to DrQA's original retriever which could have caused a decrease in accuracy.

1. Metric Based Cutoff. We used the original TF-IDF weighted bigram features and returned documents based on whether or not the document score exceeded a cutoff metric, testing at a cutoff of 150 and 200. Accuracy was close to the accuracy achieved by DrQA's original 5 document cutoff.
2. Approximate Nearest Neighbors. ANN is by definition an approximate algorithm that is less accurate than a true nearest neighbors search. We used the original TF-IDF weighted bigram features to return a number of documents (greater than the 5 DrQA originally uses), and then from those documents return a number of paragraphs based on a cosine similarity cutoff without approximate algorithms. However, while this was a substantial improvement on our original results, the accuracy still lagged significantly behind DrQA's original retriever. Based on this result, the use of ANN only accounts for part of the inaccuracy. Because of runtime constraints, attempting to query the entire corpus requires the use of an ANN library, which automatically puts it at a disadvantage. Some of the precision limitations of ANN, as alluded to previously in this report, are portrayed in figure 4.

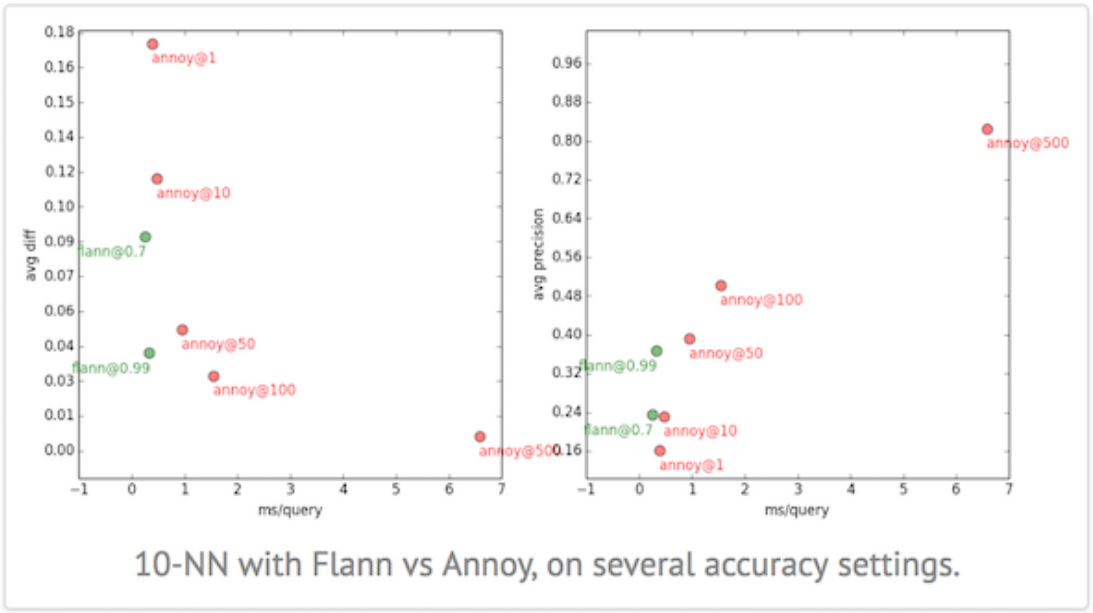

Figure 4: On the right part of this picture, we see the lack of precision of annoy for low levels of nearest neighbors, and the rate of precision increase decreases as we increase the ms/query.[5]

3. Paragraph Level Retrieval. We used the original TF-IDF weighted bigram features to return more than 5 documents, and then filter the documents according to a cosine similarity cutoff with the query. Our results were close (but still below) the accuracy of DrQA's original retriever, and substantial higher than any results using paragraph level retrieval, even with the same documents. As the set of documents we retrieve in experiments with true nearest neighbors is a strict superset of the documents the original retriever returns, we can only conclude that paragraph level retrieval tends to cause the retriever to miss the correct paragraphs, and that this is the primary cause of inaccuracy. In addition, in all our experiments which used true nearest neighbors, we saw a substantial increase (on the order of 10x) in runtime, almost certainly as a result of computing vector embeddings in the retrieval process; any potential accuracy gains from using vector embeddings would be negated by either ANN or runtime performance loss.
4. Vector Features and Cosine Similarity. Compared to DrQA's original retriever, each of our experiments which had a final stage which consisted of a cosine similarity cutoff on a vector representation of both the query and the document/paragraph text performed worse. We hypothesize that this is because the question and answer are not necessarily or even typically close; that is, the words in the query and the answer don't frequently appear in each others' context. Broad statistics suggest this hypothesis could be reasonable; the average cosine similarity between the query and the answer text itself was 0.45, which implies the cosine similarity between the query and the document/paragraph text would be even lower. This would also partially account for the accuracy loss we see using paragraph level embeddings; even though the query and the answer may not be close, a document which is similar to the query would likely contain many details about key words in the query, some of which would potentially include the answer (this is very similar to how a human might browse Wikipedia for an answer, by looking at documents relevant to the query for sections relevant to the answer). Consequently, a document level retrieval would correctly identify relevant documents; however, a paragraph level retrieval would likely ignore the paragraphs which have the answer (probably in favor of summary/document head paragraphs, which are more likely to be similar to the query).

7 Future Work and Endeavors

While our work uses novel techniques to implement and evaluate our variants of DrQA, there are multiple instances of future work that can be continued from our endeavor. First, noting that we detect cosine similarity to be fairly weak between the majority of question-answer pairs, further research could be pursued to investigate SQuAD's internal structure to examine and find attributes that cause this disconnect between cosine similarity in questions and answers. This investigation can bring out further revelations about the connection between the SQuAD dataset and question-answer vector cosine similarity. If data sets with similar structures and content to SQuAD (human-interest, general) are crafted in the future, further investigations can be performed on the attributes of these data sets as well.

An even more pressing and potentially revolutionary area of future work that can come out of this work is for researchers to perform additional evaluations on the efficacy of word embedding vectors themselves in more intricate word similarity evaluations. Currently, most word embedding vectors are highly regarded in their ability to measure cosine similarity between individual words, especially evidenced by model representations such as GloVe and Word2Vec. However, the results of this paper shed light on the potential incompleteness of word embedding models in representing question-answer type relationships. As was discussed in prior segments of the paper, humans find extremely high utility in open-domain QA tools that are able to model these relationships, and we would hope that our research in this paper can lead future researchers to further investigate and invent word-embedding models that better represent question-answer pairs. Perhaps certain embedding models in the future could be tweaked to certain NLP domain-based problems, open-domain QA being an example.

8 Acknowledgements

We would like to thank Professor Sida Wang for instructing us in this course (COS 495) and guiding our trajectory in this project. We would also like to thank Mikhail Khodak for introducing us to his summary paper of DrQA, and Mark Martinez for serving as our teaching assistant for the duration of the course.

References

[1] Mikhail Khodak. Summary and discussion of drqa, a combined system for machine reading at scale. 2018.

[2] Jason Weston Danqi Chen, Adam Fisch and Antoine Bordes. Reading wikipedia to answer open-domain questions. 2017.

[3] Approximate Nearest Neighbors in C++/Python optimized for memory usage and 2018. https://github.com/spotify/annoy loading/saving to disk. Github.

[4] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. 2016.

[5] https://rare-technologies.com/performance-shootout-of-nearest-neighbours-querying/ Radim Rehurek. Performance shootout of nearest neighbors: Querying. Accessed: 2018-05-15.