# Deep Relevance Matching for Multi-hop Question Answering

Ryan Goldenberg, Carlo Ruiz, Bryan Li
Columbia University
New York, NY
{rg3155, csr2131, bl2557}@columbia.edu

## Abstract

*HotPotQA is the first publicly available multi-hop question answering dataset. The baseline provided by HotPotQA authors has a noticeable discrepancy between the state-of-the-art QA architecture, and the rudimentary information retrieval model. We take the QA system as a black box, validate their results, then overhaul the information retrieval component of their model ensemble. We first replicate the author's preprocessing steps, from parsing a raw-text Wikipedia dump of 5 million articles, to creating a bigram TD-IDF model to prune to 100 articles for each question. We then train and run a deep relevance matching model to re-rank the candidate articles. We find that our IR model considerably improve both baseline retrieval and question answering performance.*

## 1. Introduction

Alan Turing famously posited the litmus test for a machine achieving human intelligence: a human-machine conversation that is indistinguishable from a human-human conversation. Question answering (QA) is an essential component of any conversation. QA, like most other AI fields, has seen considerable growth with the advent, or rather, the recent accessibility, of neural networks.

Building a QA agent is a two pronged problem. When faced with a question, the QA agent must first, very quickly, filter out irrelevant information. Subsequently, the agent must synthesize an answer from the resulting subset of information. A QA agent's knowledge is traditionally a set of strings or some data structure that wraps the string with relevant metadata. These pieces of information are called documents in NLP and QA literature. Single hop questions are those whose corresponding answer can be found in a single document. Multi-hop questions are those questions whose answer cannot be sourced form a single document—they require a synthesis of two or more different documents.

The filtering of irrelevant information, or, equivalently, the identification of relevant documents, is called information retrieval (IR). In IR based approaches to question answering, the system typically retrieves a small set of relevant documents out of a much larger corpus, which a downstream model analyzes to formulate an answer. The performance of the retrieval component influences the upper bound on the overall performance, as correct answers require the relevant document(s) to be in this result set. This effect is magnified in the multi-hop question answering task since the answer can only be formed by multiple relevant documents, often focused on separate topics.

This paper presents our work in quantifying the effect that information retrieval has on the multi-hop question answering task. We compare the techniques of bigram TD-IDF and a deep relevance matching network (DRMM) [2] for both IR performance, and QA performance, and find the latter approach improves significantly on the baseline. We also discuss how we have made our data preprocessing as computationally efficient as possible, given our limited resources. We have two repositories for the codebase: https://github.com/ryandgoldenberg1/hotpot_ir and https://github.com/carloruiz/MatchZoo.

## 2. Related Work

### 2.1. Question Answering

A baseline model on HotpotQA was provided by the authors. They implement the model described in Clark and Gardner (2017), which subsumes the latest technical advances in question answering, including character-level models, self-attention (Wang et al. 2017), and bi-attention (Seo et al. 2017). In contrast, the information retrieval method they use to select relevant documents over the corpus is simple. The database (further explained in section 3) consists of 5,000,000 first paragraphs from Wikipedia.

For the HotpotQA baseline model, first they encode the question (the query) and the context (the 10 articles to select from) in dual neural networks. They use GloVe for the word embeddings, as well as custom character embeddings, on top of which they apply convolutional layer (for characters only). Then they encode words and characters together

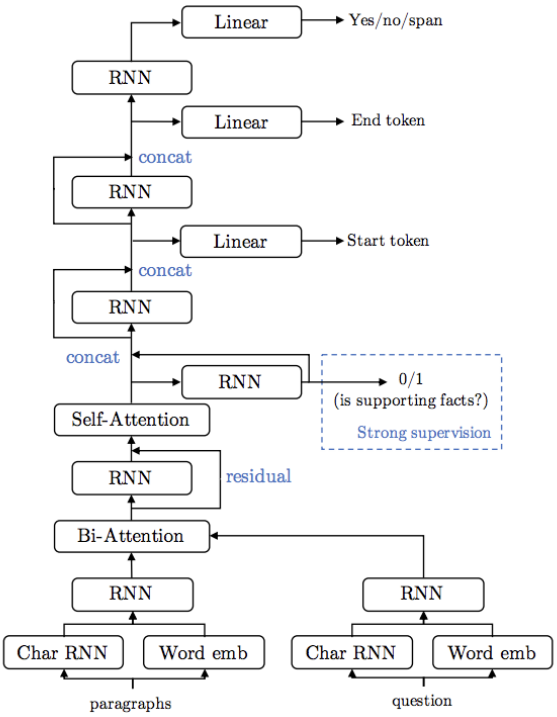


Figure 1. HotPotQA model architecture

**Paragraph A, Return to Olympus:**
[1] *Return to Olympus is the only album by the alternative rock band Malfunkshun.* [2] *It was released after the band had broken up and after lead singer Andrew Wood (later of Mother Love Bone) had died of a drug overdose in 1990.* [3] Stone Gossard, of Pearl Jam, had compiled the songs and released the album on his label, Loosegroove Records.

**Paragraph B, Mother Love Bone:**
[4] *Mother Love Bone was an American rock band that formed in Seattle, Washington in 1987.* [5] The band was active from 1987 to 1990. [6] *Frontman Andrew Wood's personality and compositions helped to catapult the group to the top of the burgeoning late 1980s/early 1990s Seattle music scene.* [7] *Wood died only days before the scheduled release of the band's debut album, "Apple", thus ending the group's hopes of success.* [8] The album was finally released a few months later.

**Q:** What was the former band of the member of Mother Love Bone who died just before the release of "Apple"?
**A:** Malfunkshun
**Supporting facts:** 1, 2, 4, 6, 7

Figure 2. Multi-hop question example. Supporting facts are highlighted in blue italics.

using a bidirectional GRU layer. The context and question encodings are then combined using bi-attention, encoded in a residual RNN layer with bidirectional GRUs, and apply a self-attention and RNN to classify supporting facts. From there, there are several RNNs (bidirectional GRU) with skip concatenation connections, and output the start token, end token, and classification of the prediction type, all trained using strong supervision.

A diagram of the model architecture from the HotpotQA paper is shown in figure 1. We take the QA model as a black box, but have experimented with hyperparamter tuning.

### 2.2. Information Retrieval

Deep relevance matching model (DRMM) is a joint deep architecture on term embeddings between pairs of terms from a query and a document [2]. This method is formulated as a relevance matching problem and is optimized for short query and long documents. Each query term's local interactions are transformed into a matching histograph, input into a feed forward matching network that learns hierarchical matching patterns and produces a matching score. The overall matching score is an aggregation of scores from constituent query terms through a term gating network.

An implementation of DRMM is provided in MatchZoo, a toolkit that aims to facilitate the designing, comparing and sharing of deep text matching models' [1]. Furthermore, a custom Keras API is provided for further configurability. However, even adapting the provided DRMM model to this dataset is tricky, and in section 5.4 we discuss this.

The relatively simple information retrieval component of HotpotQA is discussed in section 4.

## 3. Dataset

HotpotQA is a dataset designed for multihop reasoning, a task that involves reasoning over multiple documents in natural language. It is not constrained to any knowledge base or knowledge schema. Question-answer pairs are collected by a carefully constructed pipeline that presents crowd-source workers with first paragraphs of two Wikipedia articles, and based on those has them write an answer and question pair. Additionally, sentences from the article that lead to the correct answer are labeled as supporting facts – this allows for more explainable decisions.

There are two types of questions. *bridge* questions are well-explained through example: "Fodder on My Wings is an album by a singer/pianist/songwriter born in which year?" The bridge entity is the singer Nina Simone, as it links the album article to the date of year she was born in. *comparison* questions look at two different articles, for example, "Who has played for more NBA teams, Michael Jordan or Kobe Bryant?"

Figure 3 shows an example, taken from the paper, of a multi-hop question from HotpotQA.

The dataset has 112,779 examples in total, and has been divided into several subsets. Train-easy contains 18,089

single-hop examples. A state-of-the-art question answer model is trained on the remainder, and those 56,814 which are correctly answered with high confidence (based on model loss) are take as train-easy. Finally, the remaining hard examples are divided into sets 15,661 train-hard, and 7,405 of each of dev, test-distractor, and test-fullwiki.

We train on the three test sets, and report results for the 7405 examples in the dev set. The authors provide a Codalab leaderboard for evaluation on the test set. We have had some issues with the submission process, so cannot present test result at this time. However, test and dev performance are expected to be comparable.

## 4. Problem Formulation

The authors of HotPotQA provides two difficulty settings for their baseline system. In the 'distractor' setting – the upper bound of IR performance – the 2 relevant gold paragraphs are selected in addition to 8 distractor paragraphs selected using bigram tf-idf.

In the 'fullwiki' setting – the IR system we aim to beat – given a question, at most 5,000 documents are selected using an inverted-index-based filtering strategy. Then, the top 10 paragraphs are selected using bigram tf-idf. Results for these two difficulty settings diverge sharply, as shown in Table 3. The authors attribute much of this to the difficulty of retrieving both gold paragraphs. If both are not retrieved, the system can only make a guess based on what it has.

We propose a new setting called **fullwiki-drmm**, in which we combine the fullwiki setting with a DRMM to rescore the documents. This approach similarly begins with inverted-index-based filtering and bigram tf-idf to select 100 documents. We train a DRMM on a subset of the train split, and apply this to the dev set to rerank the 100 documents.

The top 10 articles as scored by **fullwiki-drmm** are used as inputs, with the original question, for the QA model. As for the QA model, we use the scripts and data files, and parameters provided by the authors to recreate their experiments as closely as possible.

### 4.1. Evaluation Criteria

We define two criteria for success: information retrieval performance, and question answering performance.

The IR metrics we use are adapted from those in the paper. With reference to [13], they are defined as:

- **MAP@10**- Mean average precision at 10 retrievals. Compute precision at rank *K* for each relevant document, and take the average precision. The MAP@10 is the mean of these over all queries. If a document is not retrieved, its term is 0.
- **Mean Rank** - Consider the rank of each relevant document. For each query, this is $.5(rank_1 + rank_2)$. If a document is not retrieved, its rank is 5001 (the number of documents retrieved by the baseline model + 1).
- **Hits@K** - Consider *K* documents, and see how many relevant documents are retrieved. 2 matches = 100% Hit@K, 1 match = 50%, 0 matches = 0%. Take the average over queries for Hits@K.

The QA metrics, with reference to [15] are:

- **EM**- Exact Match, the percentage of predictions that match any of the ground truth answers exactly.
- **F1** - the average overlap between the prediction and ground truth answer by taking both as bags of tokens and computing the F1. These F1s are then averaged for all queries.

Both EM and F1 are computed for answer, supporting fact, and joint settings. They are found per example, and averaged over all examples. The joint scores are calculated as

$$P^{(joint)} = P^{(ans)}P^{(sup)},\ R^{(joint)} = R^{(ans)}R^{(sup)}$$

$$F_1^{(joint)} = \frac{2P^{(joint)}R^{(joint)}}{P^{(joint)} + R^{(joint)}}$$

$$EM^{(joint)} = \begin{cases} 1 & EM^{(ans)} = EM^{(sup)} = 1 \\ 0 & otherwise \end{cases}$$

The baseline we will be measuring our model's performance against is the baseline model provided by the HotpotQA authors. Table 4 shows the baseline model performance, taken from the HotpotQA website [8].

## 5. Methods

### 5.1. HotPotQA Model Training and Reproduction

The authors provide a implementation of the baseline HotpotQA model, as well as JSON files for test, dev-distractor, and dev-fullwiki datasets. The format of data is a list of queries, each of which has fields 'question', 'answer', 'context', 'supporting facts', etc. 'context' contains candidate articles possibly containing the answer, and top 10 results from their IR system (shuffled order).

Running their preprocessing scripts takes a few hours, and generates vocabulary and embedding files. Then we can run the training scripts; each epoch takes 30 minutes, and learning converges in around 15 epochs (approximately 10 hour training time). The authors say that with their provided parameters the performance reaches 58% F1. Our best reproduction has 57.4% F1. We will take this model, then, as the black box for all future IR work.

We experimented with tuning dropout, as shown in table 1. Performance degrades with higher values, and we hypothesize this is because of the difficulty of the problem and sparsity of the data precluding overfitting.

| Dropout | F1 |
|---|---|
| 1.0 | 57.42% |
| 0.8 | 53.81% |
| 0.6 | 50.77% |

Table 1. HotpotQA Baseline Reproduction Performance

## 5.2. IR training data extraction

Due to the large size of the full corpus (5.5M documents, 80G compressed), we restrict ourselves to looking at the subset of the corpus contained in the combined hotpot examples (0.51M documents). We extract the text from the provided Wikipedia dump, by parsing HTML tags and truncating documents such that they contain full paragraphs and are at least 500 characters long, as described in Hotpot's methodology. It's not clear from the paper why the documents are truncated, but we suspect it's for making the query processing computationally manageable. The title and first few paragraphs of Wikipedia articles are intuitively strong relevance signals also.

## 5.3. Bigram TF-IDF Baseline

To replicate the results of the paper as a baseline, we implemented a TF-IDF bigram information retrieval model on the Wikipedia dump (see code **here** on Github).

### 5.3.1 Preprocessing

For preprocessing, we perform tokenization and stemming with the NLTK library, using Punkt sentence tokenization and the Snowball stemmer. We also remove punctuation and stopwords from the NLTK stopwords list. From the tokens, we construct a unigram and bigram vocabulary by counting the frequencies of token occurrences for efficiency. Then the inverse document frequencies are computed for each token. The last two steps are performed using PySpark's MLlib implementation.

### 5.3.2 Scoring

Our implementation of TF-IDF scoring uses all unigrams and bigrams as tokens, as the Hotpot paper does. We compute the TF-IDF score as a dot product of the TF-IDF weighted query vector and TF-weighted (bag of words) document vector. TF is the number of times the token appears in the text, and IDF is calculated as

$$IDF(t, D) = log \frac{N}{\|\{d \in D : t \in d\}\| + 1}$$

### 5.3.3 Inverted Index Retrieval

For filtering from the inverted index, we sort the matching documents by the number of distinct tokens they match and choose the top 5000. This is implemented by joining documents and queries on token IDs, and then aggregating and filtering the document query pairs. Then we score this subset using the tf-idf scoring schema above and choose the top 100, similar to the technique used by the authors. We call this setting 'fullwiki-reimp' (fullwiki reimplemented).

We use the top 100 articles, instead of top 5000, as it was both time and memory-restrictive to query 5000 times for each query on even the .51M subset of the corpus. Parallelizing the queries on a GCP machine with 56 CPUs and 624 GB of memory, we are able to perform 2000 queries/hr. In total, we retrieved 7500 queries of the train set, and all 7405 queries of the dev set.

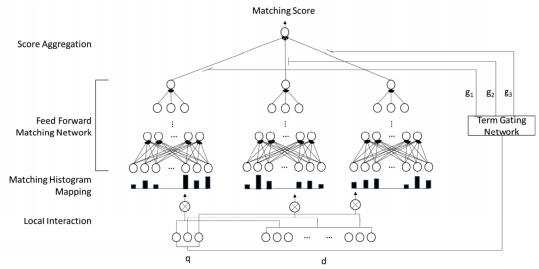


Figure 3. DRMM model architecture

## 5.4. DRMM and MatchZoo

Historically, IR has been formulated as a textual or semantic matching problem. The novelty in DRMM is that it frames IR as a ranking problem. First, we embed the whole corpus using some pre-trained word embedding (we used GLoVe 300d embedding). For each query term, DRMM computes the cosine similarity ([-1, 1]) with each document term. The range [-1,1] is partitioned into 60 bins of equal width. A histogram is then built by counting the number of cosine similarity values falling withing each bin. This histogram is represented with a 60 dimensional vector. Training involves computing the histograms between the query and and 2 documents. These histogram vectors are then fed into a fully connected network. This architecture is shown in figure 3.

In order to use the MatchZoo models we had to reformat HotpotQA dataset into MatchZoo format. Beside some basic formatting (stemming, removing stop words, etc.), this involved using pretrained GloVe 300d embeddings and computing the histograms necessary for DRMM. We achieved some promising results when doing IR only on the 10 documents already provided in HotpotQA. WikiQA, a related single-hop wiki QA dataset, was used to initialize the DRMM training [14].

However, in order to close the gap between fullwiki and distractor, we need to do IR on the whole document corpus of 5 million documents. In order to reduce the search space,

| Setting | MAP | Mean Rank | Hits@2 | Hits@10 |
|---|---|---|---|---|
| dev fullwiki | 43.93 | 314.71 | 39.43 | 56.06 |
| test fullwiki | 43.21 | 314.05 | 38.67 | 55.88 |

Table 2. Retrieval performance reported by HotpotQA

| Setting | MAP@10 | Mean Rank | Hits@2 | Hits@10 | Hits@100 |
|---|---|---|---|---|---|
| distractor | 37.34 | 5.49 | 20.15 | 100.00 | N/A |
| fullwiki | 18.10 | 2218.33 | 11.35 | 55.73 | N/A |
| **fw-reimp** | 38.34 | 1072.04 | 32.47 | 56.64 | 78.75 |
| **fw-drmm** | 52.19 | 1018.00 | 45.80 | 66.30 | 78.75 |

Table 3. Retrieval performance on dev set with our model

| dataset | $EM^{(ans)}$ | $F_1^{(ans)}$ | $EM^{(sup)}$ | $F_1^{(sup)}$ | $EM^{(jnt)}$ | $F_1^{(jnt)}$ |
|---|---|---|---|---|---|---|
| distractor | 45.60 | 59.02 | 20.32 | 64.49 | 10.83 | 40.16 |
| fullwiki | 23.95 | 32.89 | 3.86 | 37.71 | 1.85 | 16.15 |

Table 4. QA Performance reported by HotpotQA

| dataset | $EM^{(ans)}$ | $F_1^{(ans)}$ | $EM^{(sup)}$ | $F_1^{(sup)}$ | $EM^{(jnt)}$ | $F_1^{(jnt)}$ |
|---|---|---|---|---|---|---|
| distractor | 43.24 | 57.20 | 20.40 | 66.07 | 10.56 | 40.00 |
| fullwiki | 23.44 | 32.81 | 5.08 | 40.59 | 2.74 | 17.03 |
| **fw-reimp** | 20.46 | 31.02 | 3.69 | 37.87 | 1.62 | 15.48 |
| **fw-drmm** | 24.69 | 36.10 | 3.95 | 41.60 | 0.75 | 17.92 |

Table 5. QA Performance with our model

we identified some 100 candidate passages per query using inverted-index filtering, then applied DRMM to rank these.

As mentioned in our proposal, MatchZoo provides implementations of other text-match methods as well as an extended Keras API to configure custom ones. We anticipate that after doing all of the processing and formatting work for DRMM, that it would be relatively straightforward to adapt to methods like DUET or DRMM-TKS.

### 5.5. Bridging between Data Formats

The output of our TD-IDF system is a JSON with outputs on each line corresponding to ID of 1 retrieved document, ID of the associated question, TD-IDF score, and rank. Using a python lookup dictionary mapping article titles to article title + content, we replace the 'context' field of the provided dev JSON (either, as the only difference between distractor and fullwiki is the 'context' field) with 100 articles for each question. This gives us a JSON file for **fullwiki-reimp** to use on the evaluation scripts.

The output of the MatchZoo DRMM model contains the same fields in a CSV file. The question IDs are the same, but the document IDs here are hashes of the document titles. We map these hash IDs to the original article IDs to retrieve titles + contents as above. This gives us a JSON file for **fullwiki-drmm**.

## 6. Results

### 6.1. IR Performance

Table 3 compares IR results for distractor and full wiki to our two models of fullwiki-reimplemented, and fullwiki-drmm. Using the given data files, we verify that dev distractor has 100% hits@10, and fullwiki has about 56%.

Though we implement other metrics besides hits@10, we cannot directly compare to the presented results as for:

- **MAP@10**- their metric is MAP@5000, but it is costly to retrieve that many articles.
- **Mean Rank** - we have 10 articles (not 5000), so there are many more unretrieved gold articles, which have rank of 5001 (as in the paper).
- **Hits@2** - the provided articles are shuffled and not in retrieval order.
- **Hits@100** - only the top 10 results are provided.

Another IR metric we have considered is normalized discounted cumulative gain at 10 [6]. Relevance for a passage is defined here to be 0 or 1 depending on whether it is a golden passage for the query. This is what MatchZoo provides by default, but we do not report here for brevity.

Looking at the most salient metric hits@10, we see fullwiki and fullwiki-reimp are similar, which confirms our correct implementation of TD-IDF. fullwiki-drmm performs 10% absolutely better (18% relatively). fw-drmm excels in all other ranks, as relative to fw-reimp it has a 14% absolutely better MAP@10, a 54 lower mean rank, and a 13.5% absolutely better Hits@2. All this shows that our inverted-index filtering to DRMM reranking among the top 100 strategy was indeed effective.

### 6.2. QA Results

Table 5 shows the performance with our reimplemented QA model (QA-reimp), which is described in section 5.1. All results use 10 articles for context, and are reported on the dev split. Comparing distractor and fullwiki settings with those of table 4, we see that our reimplementation just about replicates the paper's results. QA-reimp does slightly better with supporting facts, and worse with answers. The difference can be attributed to better hyperparameter tuning on their end.

**fullwiki-reimp**, despite having a comparable Hits@10, does a bit worse on all metrics than fullwiki. We attribute this to a difference in the text of retrieved articles. As mentioned in 5.2, the authors say they limit articles to about 500 characters. However, inspection of the provided data files revealed that most articles had between 400 and 900 characters. Tuning this as another hyperparameter, we found a max of 850 characters to work the best.

Despite this differential, we find that **fullwiki-drmm** performs better than fullwiki in all measures except for joint EM, with a 1.25% absolute increase in answer EM and 3.3%

absolute increase in answer F1. These are about 10% relative increases, and considering the 18% relative increase in Hits@10, is in line with what is expected.

### 6.3. Analysis of an Exmaple

In this sections, we look at an example from the dev set in the fullwiki and fullwiki-drmm settings. We bring up the bridge question from section 3, "Fodder on My Wings is an album by a singer/pianist/songwriter born in which year?" The top 10 results for fullwiki are:
['Rouba', 'Moriah Peters', 'Javier Limn', 'Jack Starr', 'Manu Manzo', 'Jessie Baylin', "Matt O'Ree", 'Dara Maclean', "O'Chi Brown", 'Fodder on My Wings']

Whereas for fullwiki-drmm they are:
['Fodder on My Wings', 'Nina Simone in Concert', 'Folksy Nina', 'Nina at the Village Gate', 'London Town (Wings album)', 'Wings discography', 'Beware My Love', 'Wings at the Speed of Sound', 'Wings over America', "She's My Baby (Wings song)"]

We observe how the the TD-IDF ranking latches onto the terms 'singer/pianist/songwriter' and returns 9 musicians, with the album at rank 10. However, fullwiki-drmm correctly reranks the 100 candidate TD-IDF articles such that the album is rank 1. The bridge entity itself, 'Nina Simone,' is not found, but the top 3 are all her albums.

Neither setting arrives at the correct answer of '1933', since multi-hop reasoning cannot be made (1972 fullwiki, 1976 fullwiki-drmm). Still, we can reasonably expect that if the bridge entity existed (perhaps if 5000 articles were retrieved) , our system would be able to rerank it so that it would occur early on in the top matches.

## 7. Responsibilities

Between the three group members in the project, the work was divided up as follows:

Ryan was responsible for implementing the bigram TD-IDF system, performing the queries, and running the black-box HotpotQA QA system. He also maintained the GCP instance and storage bucket system.

Carlo was the MatchZoo and deep-relevance matching model (DRMM) expert, and set up the scripts for training these new IR systems and extracting queries.

Bryan was responsible for writing all the code bridging between the data formats, helping out Ryan with reimplementation, and running the HotpotQA evaluation scripts.

All of us contributed to writing reports, with Bryan particularly responsible for making sure the prose is cohesive, all sections are filled out, and all questions are answered.

## 8. Development Environment

We maintained an active messaging group, and met up weekly to share individual progress, and sync up the different parts of the project. We decided it made sense to use two Github repositories, and both started as forks of bigger projects, HotpotQA for QA and MatchZoo for IR, each with many dependencies.

Our runtime environment is hosted on Google Cloud Platform. Each Compute Engine machine varies in CPUs, memory, GPUs, etc. This allows us to only use as many resources as required for a task. To transfer files between instances, we use the integrated Google Cloud Storage, a bucket data store.

Our Baseline + TD-IDF Github repo has 4000+ lines of code and 27 commits. Our fork of MatchZoo has 4 commits with about 400 lines of code.

## 9. Conclusions and Future Work

We introduce the **fullwiki-drmm** setting for HotpotQA. This first does an inverted-index based filtering strategy to prune the full wiki dump of 5 million articles to 100, then reranks those 100 using a deep relevance matching model. We find that this noticeably improves on IR metrics, with an 18% relative increase in Hits@10. Small but significant increases in QA metrics are seen as well, with 10% relative increases in both answer EM and F1.

In order to be able to improve on the HotpotQA baseline, we had to rerun their QA model and reimplement some of their methods. Training scripts, and datasets for the train, dev-distractor, and dev-fullwiki settings were provided. Once trained, the QA model was taken as a black box. Reimplementing the full inverted-index based filtering strategy proved impossible with our resources, so we used a 0.51M subset of the full wiki dump, and retrieved the top 100 articles. We also reimplemented the bigram TF-IDF baseline, terming this **fullwiki-reimp**, and found performed roughly the same as fullwiki.

Future work includes running our approaches on the full dump of 5.5M articles, and retrieving 5000 articles for each query as the paper did. We expect that are gains would be magnified given this huge increase in the amount of training data, which we know to greatly assist in deep learning. We can ask the authors of the paper through email for the 5000 context articles for each paper to ease the computational requirements. With this, training a new MatchZoo DRMM model should be just a matter of time. Also, MatchZoo allows for configuration of several other text matching models – DUET and DRMM-TKS, to name a few.

We also hope to submit our predictions on the test set to join the HotpotQA leaderboard. As of 5 Dec 2018, we see that 1 group has already beaten the baseline by a good amount. We suspect that most of these gains were made in upgrades to the QA system, and would like to see if and how their work modified IR. Multi-hop reasoning is a worthwhile field for further research, as it pushes the boundaries of how machine reasoning.

## References


[1] Fan et al. (2017). *MatchZoo: A Toolkit for Deep Text Matching*, https://arxiv.org/pdf/1707.07270.pdf

[2] Guo et al. (2016). *A Deep Relevance Matching Model for Ad-hoc Retrieval*, http://www.bigdatalab.ac.cn/ gjf/papers/2016/CIKM 2016a_guo.pdf

[3] Mitra et al. (2016). *Learning to Match Using Local and Distributed Representations of Text for Web Search*, https://arxiv.org/pdf/1610.08136.pdf

[4] Pang et. el. (2016), *Text Matching as Image Recognition*, https://arxiv.org/pdf/1602.06359.pdf

[5] Pang et. el. (2017) *DeepRank: A New Deep Architecture for Relevance Ranking in Information Retrieval*, https://arxiv.org/pdf/1710.05649.pdf

[6] Wikipedia, *Discounted cumulative gain*, https://en.wikipedia.org/wiki/Discounted_cumulative_gain

[7] Yang et. al. (2018) *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering*, https://nlp.stanford.edu/pubs/yang2018hotpotqa.pdf

[8] Yang et. al. (2018), *HotpotQA Homepage*, https://hotpotqa.github.io/

[9] Quora (2017), *First Quora Dataset Release: Question Pairs*, https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs

[10] Wikipedia, *Okapi BM25*, https://en.wikipedia.org/wiki/Okapi_BM25

[11] Kratzwald et. al., *Adaptive Document Retrieval for Deep Question Answering*, https://arxiv.org/pdf/1808.06528.pdf

[12] Clarke et. al., *Novelty and Diversity in Information Retrieval Evaluation*, https://plg.uwaterloo.ca/ gvcormac/novelty.pdf

[13] Doug Oard, Ellen Voorhees., *Introduction to Information Retrieval: Evaluation*, https://web.stanford.edu/class/cs276/handouts/EvaluationNew-handout-1-per.pdf

[14] Yi Yang, Wen-Tau Yih, Christopher Meek. *WikiQA: A Challenge Dataset for Open-Domain Question Answering*, https://aclweb.org/anthology/D15-1237

[15] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. *Squad: 100,000+ questions for machine comprehension of text*, http://www.aclweb.org/anthology/D16-1264