Ranking Answers and Web Passages for Non-factoid Question Answering: Emory University at TREC LiveQA

Denis Savenkov
Emory University
dsavenk@emory.edu

Abstract

This paper describes a question answering system built by a team from Emory University to participate in TREC LiveQA'15 shared task. The goal of this task was to automatically answer questions posted to Yahoo! Answers community question answering website in real-time. My system combines candidates extracted from answers to similar questions previously posted to Yahoo! Answers and web passages from documents retrieved using web search. The candidates are ranked by a trained linear model and the top candidate is returned as the final answer. The ranking model is trained on question and answer (QnA) pairs from Yahoo! Answers archive using pairwise ranking criterion. Candidates are represented with a set of features, which includes statistics about candidate text, question term matches and retrieval scores, associations between question and candidate text terms and the score returned by a Long Short-Term Memory (LSTM) neural network model. Our system ranked top 5 by answer precision, and took 7th place according to the average answer score. In this paper I will describe our approach in detail, present the results and analysis of the system.

1 Introduction

Over the years of question answering research the focus was mainly around factoid questions, which constitute only a small part of user information needs. Factoid questions are usually defined as those that can be answered with a short phrase, e.g. a number or a named entity. Advice, opinion, recommendation, manner, instruction and other similar types of questions are beyond the scope of the factoid question answering systems. One way to solve these questions is to ask other people, and community question answering websites (e.g. Yahoo! Answers[1], Answer.com, StackExchange.com, etc.) became very popular and currently contain millions of questions and answers from real users. In 2015 TREC started a series of LiveQA evaluation campaigns, that targets automatic answering of questions posted to Yahoo! Answers in real time. The majority of such questions can be classified as non-factoid. Previously, research in non-factoid question answering often focused on re-ranking of answers already present in a CQA archive [1]. LiveQA opens up new opportunities, as it allows a system to use any existing resources, not just a fixed collection of CQA QnA pairs or documents.

The system I developed builds on some existing research on question answering and is based on a combination of CQA archive and web search based approaches. There are a lot of different types of questions that users post to CQA websites and it is probably beneficial to study them separately. However, for simplicity the model I built treats all questions in the same way. My system is based on a single trained model, that ranks a set of extracted answer candidates and returns the top one as the response. Preliminary analysis of questions and potential answer sources gave an insight that the best data source is answers to similar questions in case they exist and we can find them. People often have similar tasks and situations which pose same questions. Therefore, it's frequently the case that a similar question was already asked by someone and potentially even received a good reply and can be reused to answer new questions [2]. Of course, many questions or their details are unique, which makes it impossible to find a good match from the existing answers. Therefore I also use web search to generate additional answer candidates. For non-factoid questions it's harder to use the redundancy of the information on the web, which is exploited very effectively in factoid QA [3]. The system

[1] http://answers.yahoo.com/

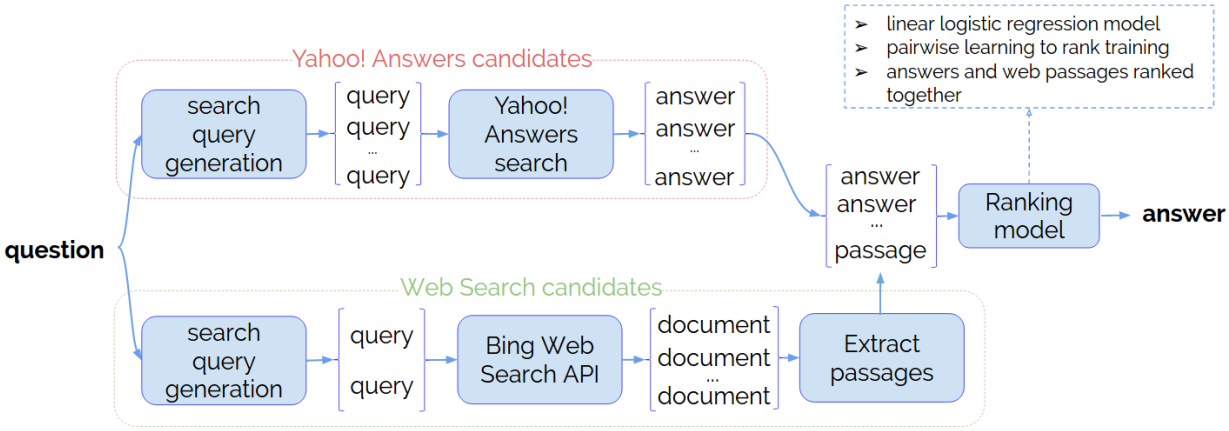


Figure 1: Architecture of our question answering system

I developed extracts passages containing question terms from all retrieved web documents independently. For training I used the publicly available collection of QnA pairs from Yahoo! Answers. The assumption made was that for each question the answer selected as the "best answer" on Yahoo! Answers is indeed the best and should be ranked higher than answers to other questions. However, taking all other answers is intractable and probably detrimental as almost all of them would be totally unrelated to the subject of the given question. Therefore, I used search to retrieve a set of similar questions and took their answers as negative examples. The following chapters describe the QA system in more detail.

2 Approach

The general architecture of our question answering system is presented on Figure 1. It uses two primary data sources for generating answer candidates: answers to similar questions from Yahoo! Answers website and documents retrieved using web search. All candidates are mixed together, ranked and the top answer is returned.

2.1 Candidate generation

Each question issued to a QA system in TREC LiveQA consists of 3 main parts: title, body and category. For example:

> **Question category**: Astronomy & Space
> **Question title**: Why do people claim the Earth isn't the center of the universe?
> **Question body**: Clearly the sun and moon are moving around the Earth otherwise we wouldn't have night and day.

When the QA system receives a question it first generates a set of candidate answers from Yahoo! Answers and regular web search. To generate a set of candidates the system produces several search queries and issues them to both resources.

To find similar questions and extract the corresponding answers from Yahoo! Answers we use the search functionality already available on the website. Some questions are very concise while other provide many useful as well as redundant details. Ideally we want to match as many of them as possible, however, there is a chance that search won't return any results if there are no good matches. Therefore the system generates a set of search queries of different granularity, issues them all to Yahoo! Answers search and collects top 10 responses from all of them. Here is the list of queries that our system generates:

- Concatenation of question title and question body (with and without stopwords)
- Question title only (with and without stopwords)

- Question title concatenated with question body and question category
- Question title concatenated with the name of the question category
- Top 5 terms from question title scored by tf-idf[2]
- Top 5 terms from question title and body scored by tf-idf

For each query and top-10 retrieved questions the system extracts its top answer if provided and puts it into the candidate pool along with some information about the corresponding question and its category.

To extract candidate passages from relevant web documents previous research in factoid question answering have tried query reformulations [4] to better match the potential answer text. However recently [5] demonstrated that such reformulations are no longer necessary as search engines have improved the query processing techniques. Inspired by this observation and considering that retrieving web documents and extracting passages from them is more time consuming, the system issues only 2 web search queries: question title and title concatenated with body. I used Bing Web Search API[3] and the system downloads top-10 retrieved documents, parses HTML code and extracts the main content text [6]. Document content is further split into sentences [7] and candidates are built by taking contiguous sequences of sentences no longer than the answer character limit[4]. The model only keeps passages that contain at least one non-stopword from the question. Web search snippets are also included as candidates.

2.2 Candidate ranking

A trained linear logistic regression model is used to rank candidate answers, represented with a set of features:

- answer text statistics: length in character, tokens and sentences, average number of tokens per sentence.
- Okapi BM25 scores, which consider question title and concatenation of title and body as queries. Term statistics were computed on Yahoo! Answers WebScope dataset. The score is calculated as follows:

$$\text{score}(A, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, A) \cdot (k_1 + 1)}{f(q_i, A) + k_1 \cdot (1 - b + b \cdot \frac{|A|}{\text{avg\_al}})}$$

  where $f(q_i, A)$ is frequency of term q_i in the answer text, $k_1 = 1.2$, $B = 0.75$ and $avg_al = 50$ (average answer length).
- term matches features: lemmas, part of speech tags of matched terms between the question and answer texts, the fraction of unique question terms matched in the answer, length of the maximum span of matched terms in the answer.
- number of matched terms between the question title, body and the title of the page from which the candidate answer is retrieved. For Yahoo! Answers the text of the retrieved question is used as title.
- category match feature for Yahoo! Answers candidates.
- pairs of lemmas from question and answer texts, that are supposed to bridge the lexical gap between question and answer language.
- average, minimum and maximum normalized pointwise mutual information (NPMI) scores between pairs of terms from the question and answer texts. The scores are estimated from QnA pairs from Yahoo! Answers WebScope dataset using the following formula:

$$\text{npmi}(q_i; a_j) = \frac{pmi(q_i; a_j)}{-\log p(q_i, a_j)}$$

$$\text{pmi}(q_i; a_j) = \log \frac{p(q_i, a_j)}{p(q_i)p(a_j)} = \log \frac{p(a_j|q_i)}{p(a_j)}$$

- QnA pair score from a Long Short Term Memory (LSTM) neural network model, described in Section 2.3.1.

[2]Document frequency is computed on WebScope collection of QnA pairs from Yahoo! Answers
[3]http://datamarket.azure.com/dataset/bing/searchweb
[4]In the final run the limit was 1000 characters

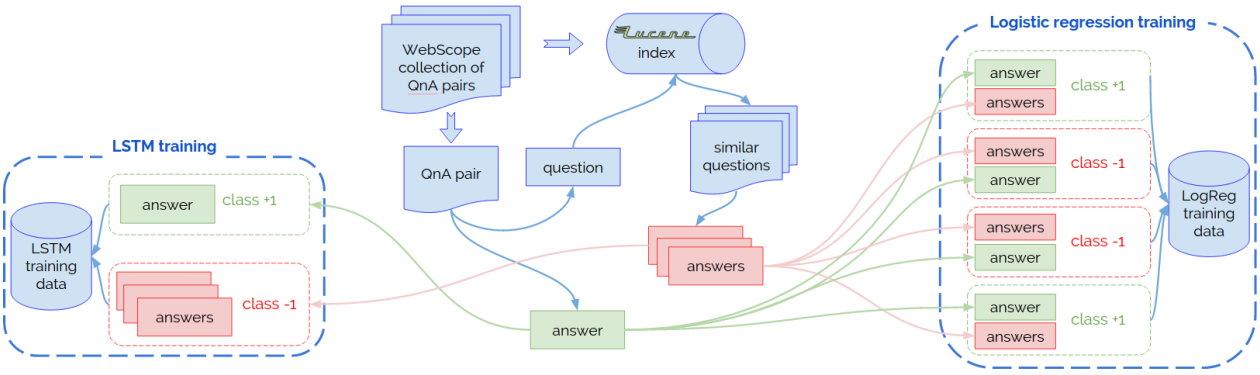


Figure 2: Workflow for generating training datasets for LSTM and answer ranking logistic regression model from the Yahoo! Answers QnA pairs

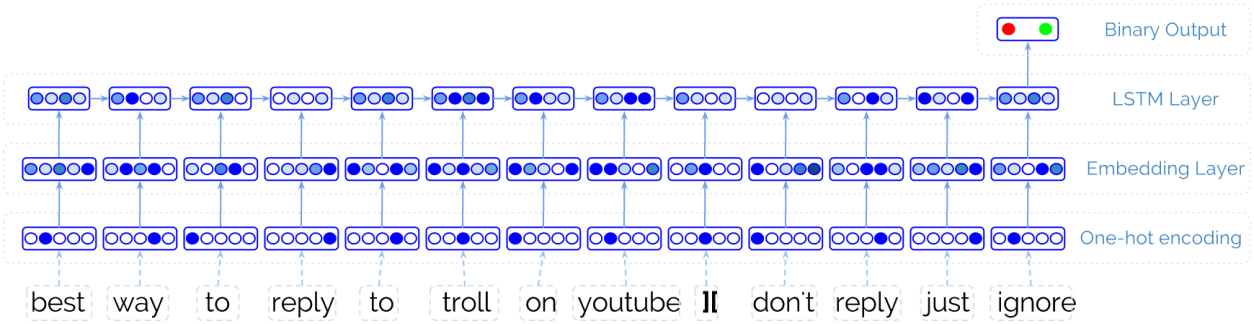


Figure 3: LSTM model for answer scoring. The example shows a QnA pair where the question is "Best way to reply to trolls on youtube?" and the answer is "Don't reply, just ignore".

The candidate with the highest score is returned as the answer to the question. If something goes wrong and no candidates were generated or some problem occurred the system returns "I don't know" as the default answer.

2.3 Model Training

There are two trained models used in the system: LSTM recurrent neural network based model, which is used as one of the features for the final logistic regression model that scores all candidates and selects the best one as the answer. I use WebScope Yahoo! Answers dataset[5] (different splits are used) to generate training data for both LSTM and ranking model, Figure 2 describes the steps I took to build training datasets.

2.3.1 LSTM model

Deep learning models had a huge success in image and speech problems and showed very promising results in natural language processing and question answering, e.g. [8, 9] to name a few. I decided to explore this direction and built a recurrent neural network model to score how well a candidate answers a question. Long Short-Term Memory (LSTM) [10] is a particular architecture of recurrent neural networks that helps with the exploding and vanishing gradients problems. The model I developed reads question and answer tokens and produces a probability score based on a vector representation of a QnA pair. Figure 3 shows the structure of the model.

Question (title with body) and answer texts are tokenized, punctuation characters are removed and for each token lowercase lemma is taken. The sequences are limited to 100 elements and concatenated through a sentinel separator character so the model could learn where the question ends and the answer starts. The

[5] https://webscope.sandbox.yahoo.com/catalog.php?datatype=l

hidden state of the model after the whole sequence is processed is used by logistic regression unit to output a probability, that a candidate answers the question well.

To train the model QnA pairs from Yahoo! Answers WebScope dataset were used (we selected a subset of questions from the categories chosen for TREC LiveQA). Each question and the corresponding best answer was used as a positive training example. Random negative examples would be too unrelated to the current question, therefore I chose to use answers to similar questions only. All QnA pairs were indexed with Lucene[6] and similar questions were retrieved using the built-in BM25 retrieval model. For each question and correct answer pair from the dataset 10 similar questions were retrieved and the corresponding answers were used as negative examples for training[7].

The model was implemented using Keras[8] library. I used an embedding and hidden layers of dimension 128 and the vocabulary size of 1M words. The model was trained using Adam optimization technique [11] with mini batches of 200 instances for 100 epochs.

2.3.2 Logistic regression model

The final model that ranks all answer candidates is a linear L2-regularized logistic regression model. To train the model we used a different split of QnA pairs from Yahoo! Answers WebScope dataset. For each question the corresponding "best answer" is taken as the correct one. To get a sample of negative examples Lucene index is used again and answers to 10 most similar questions are retrieved. Different from LSTM model training, here I took a pairwise approach for learning to rank and generated training examples from pairs of different answers to the same question, where one answer is the correct one. That is, let the current question be Q, its "correct" answer A^*, and retrieved candidates $A_1, ..., A_n$. Each candidate is represented with a set of features: $f(Q, A^*)$, $f(Q, A_1)$, ..., $f(Q, A_n)$. For each $i = 1..n$ we create two training instances, i.e. class 1: $\langle A^*, A_i \rangle$ and class -1: $\langle A_i, A^* \rangle$. Each such instance is represented with pairwise differences of features, e.g. $\langle A^*, A_i \rangle : f_{pair}(Q, \langle A^*, A_i \rangle) = f(Q, A^*) - f(Q, A_i)$. The trained model is linear, therefore if $w(f(Q, A^*) - f(Q, A_i)) > 0$ then $wf(Q, A^*) > wf(Q, A_i)$ and we can rank candidates by the score produced by the model, i.e. $wf(Q, A_i)$.

3 Evaluation

From the final run of the system, 1087 questions were judged by the organizers on a scale from 1 to 4:
4: Excellent - a significant amount of useful information, fully answers the question
3: Good - partially answers the question
2: Fair - marginally useful information
1: Bad contains no useful information for the question
-2: the answer is unreadable (only 15 answers from all runs were judged as unreadable)

The following performance metrics were reported:

- **avg-score(0-3)**: average score over all questions, where scores are translated to 0-3 range. This metric considers "Bad", unreadable answers and unanswered questions as having score 0
- **succ@i+**: the fraction of answers with score i or greater (i=1..4)
- **p@i+**: the number of questions with score i or greater (i=2..4) divided by the number of answered questions

Table 1 provides the results of top 5 teams by average answer score, results for our system and average scores. Please refer to the [12] for more details and results of all systems.

The absolute values of the performance metrics demonstrate a great room for improvement as our system was able to return partial or good answer only in 23% of the cases (prec@3+) when the answer was returned. And for 60% of the questions the answer doesn't contain any useful information.

[6] https://lucene.apache.org/
[7] It's true, that some of them can indeed be relevant to the original question
[8] http://keras.io

Table 1: Results of the TREC LiveQA evaluation of top 5 systems, Emory University QA system and average results of all systems. ↑ means that results of Emory system in this metric are above average, and ↓ means that results are below average

	# answers	avg score (0-3)	succ@2+	succ@3+	succ@4+	p@2+	p@3+	p@4+
1. CMUOAQA	1064	1.081	0.532	0.359	0.190	0.543	0.367	0.179
2. ecnucs	994	0.677	0.367	0.224	0.086	0.401	0.245	0.094
3. NUDTMDP1	1041	0.670	0.353	0.210	0.107	0.369	0.219	0.111
4. RMIT0	1074	0.666	0.364	0.220	0.082	0.369	0.223	0.083
5. Yahoo-Exp1	647	0.626	0.320	0.211	0.095	0.538	0.354	0.159
7. Emory	884↓	0.608↑	0.332↑	0.190↑	0.086↑	0.408↑	0.233↑	0.106↑
Average results	1007	0.467	0.262	0.146	0.060	0.284	0.159	0.065

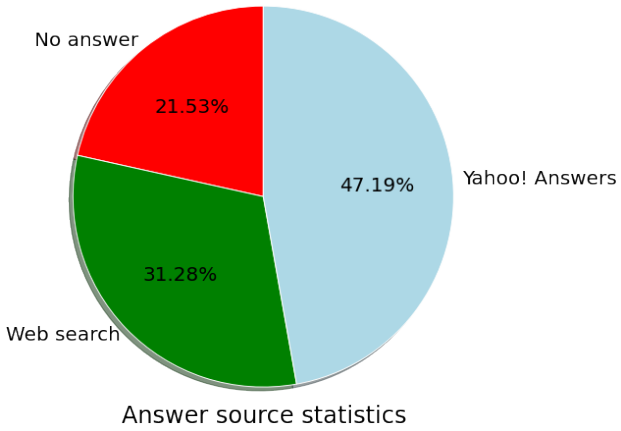


Figure 4: Distribution of sources for answers returned by our system

4 Analysis

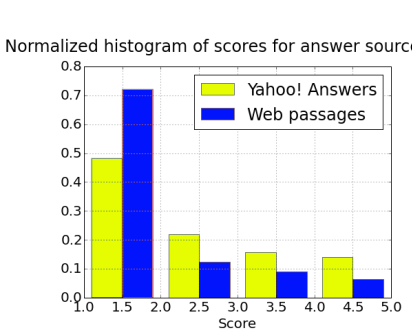
In this section we will answer some of the questions about the performance of different system components and their relative importance.

The first question, that we are going to study is the relative effectiveness of web passages and answers to previously posted questions for answering new questions. As Figure 4 shows, almost half of all answers returned by my system were generated from Yahoo! Answers. In ∼21% of the cases our system didn't return any results[9], and in the rest ∼31% of the cases a passage from a web page was returned as the answer. I further looked into the domains of the web pages used to generate the answer and noticed, that many more were extracted from other community question answering websites and forums.

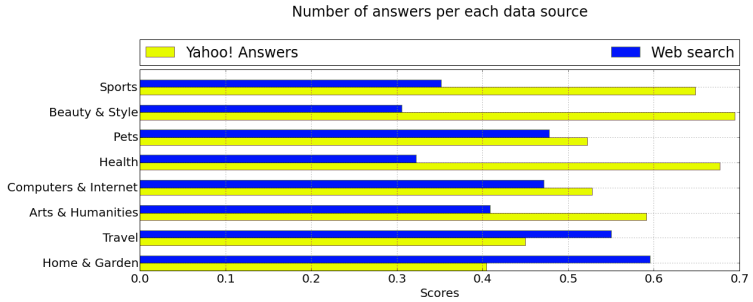
The quality of answers generated from passages built from web search results are lower on average compared to Yahoo! Answers candidates. Figure 5 shows the distribution of scores for each of our data sources. Some categories were harder than the other [12] and as we see on Figure 5b in some cases web passages were actually more effective than answers to previously posted questions.

The next question, that we analyze is the effectiveness of search query generation strategies. Figure 6 plots average number of candidates and the position of the best candidate retrieved by each of the question generation strategies. The longer the search query the less results it retrieved, which is expected, and the lower the quality of the candidates. As a result, in half of the cases the answer returned by our system was retrieved using just top 5 highest IDF terms as the query[10]. For web search we only used 2 query generation strategies, namely question title and concatenation of title with body. Analogously, concatenation of title

[9]This happened mainly due to a couple of technical issues that made our system unresponsive for quite some time
[10]The same candidate is often also retrieved by other queries

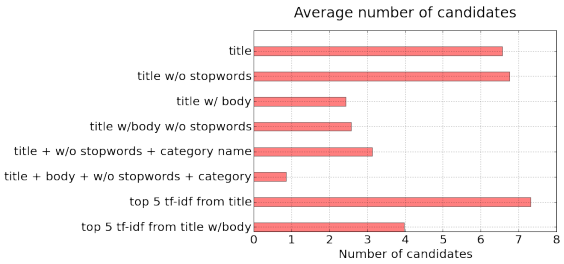


(a) Histogram of qrel scores

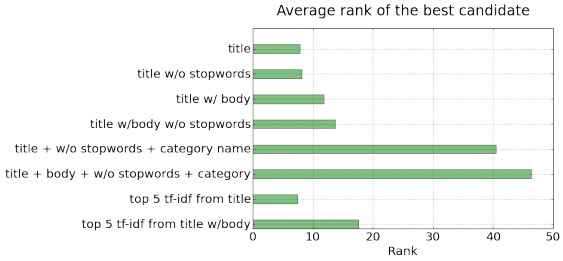


(b) Average scores of answers from Yahoo! Answers and web passages for different categories

Figure 5: Comparison of web passages and Yahoo! Answers as candidate sources



(a) Average number of candidates



(b) Average rank (according to our ranking model) of the best candidate

Figure 6: Comparison of different query generation strategies for Yahoo! Answers similar questions search

with body query had lower quality and more often returned few or no results.

Figure 7 demonstrates a plot of importances of different features in our answer ranking linear logistic regression model. The feature with the highest weight is category match, but we should note, that this feature is overfitted to the way we build training set (category of the correct answer always matched the category of the question). The next most useful feature is the cosine similarity between the page title (or question text for Yahoo! Answers) and the current question, followed by BM25 score, number of matched verbs, etc.

I also looked through a small sample of our answers manually. There are a number of typical problems, and one of them is the lack of good question semantic similarity measure. E.g. the question *"Is there a section fro the subject of writing"* in the `Books & Authors` category retrieved a question *"I can't write in the To: Cc: Subject: section"* from the `Yahoo Mail` category. Even though the questions have many terms

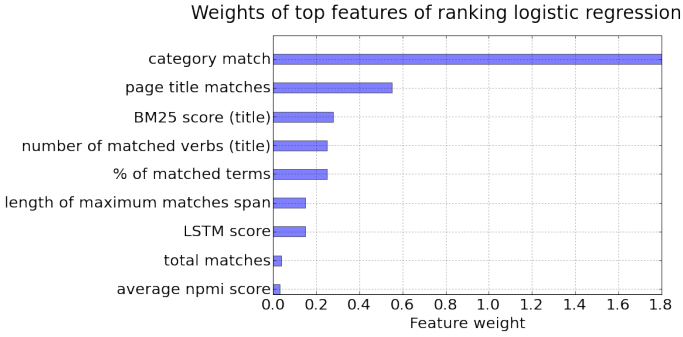


Figure 7: Weights of features in answer ranking logistic regression model

in common, they are obviously semantically unrelated. Therefore, in future we need to focus more on better question similarity measures.

Answer doesn't have to have many words in common with the question. On the contrary, the maximum possible term overlap will be if a candidate is just a copy of the answer. This was one of the problems for answers, retrieved from the web search results. The way we used to generate the training data didn't include such "artificial" cases, however, they are pretty common in practice. For example, in a number of cases the answer our system chose came from a forum post and instead of selecting the answer posts, the system ranked the question post higher as it had more term matches. The winning CMU team addressed this issue by considering answer-clue pairs, where the clue is supposed to match the question text and the former answers the question. We plan to explore a similar strategy.

5 Conclusion

The pilot year of TREC LiveQA establishes a very good baseline for the future of non-factoid question answering. It confirmed that the task itself is quite challenging as only ∼35% of the questions returned by the winning system had a score of 3 or higher, and there is still a big gap between in the quality of human and machine answers. It will be exciting to see the next version of the task next year, and how the participants will build on this year approaches.

References

[1] Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. Learning to rank answers to non-factoid questions from web collections. *Computational Linguistics*, 37(2):351–383, 2011.

[2] Anna Shtok, Gideon Dror, Yoelle Maarek, and Idan Szpektor. Learning from the past: Answering new questions with past answers. WWW '12, pages 759–768, New York, NY, USA, 2012. ACM.

[3] Jimmy Lin. An exploration of the principles underlying redundancy-based factoid question answering. *ACM Trans. Inf. Syst.*, 25(2), April 2007.

[4] Eugene Agichtein, Steve Lawrence, and Luis Gravano. Learning search engine specific query transformations for question answering. WWW '01, pages 169–178, New York, NY, USA, 2001. ACM.

[5] Chen-Tse Tsai, Wen tau Yih, and Christopher J.C. Burges. Web-based question answering: Revisiting askmsr. Technical Report MSR-TR-2015-20, April 2015.

[6] Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 441–450, New York, NY, USA, 2010. ACM.

[7] Christopher D Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J Bethard, and David McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of ACL*, pages 55–60, 2014.

[8] Lei Yu, Karl Moritz Hermann, Phil Blunsom, and Stephen Pulman. Deep learning for answer sentence selection. *arXiv preprint arXiv:1412.1632*, 2014.

[9] Di Wang and Eric Nyberg. A long short-term memory model for answer sentence selection in question answering. *ACL*, 2015.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Eugene Agichten, David Carmel, Donna Harman, Dan Pelleg, and Yuval Pinter. Overview of the trec 2015 liveqa track. In *Proceedings of Text Retrieval Conference*, 2015.