ICTNET at Trec 2019 Conversational Assistance Track

Changying Hao[1,2], Yuanyuan Zhang[1,2], Weifeng Yang[1,2], Heng Zhao[1,2]
[1]CAS Key Lab of Network Data Science and Technology,
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2]University of Chinese Academy of Sciences, Beijing ,China
{haochangying,zhangyuanyuan,yangweifeng,zhaoheng}18s@ict.ac.cn

Abstract

In this paper we report on our participation in the TREC 2019 Conversational Assistance Track which focuses on Conversational Information Seeking（CIS）namely understand the dialogue context and retrieve candidate response information from collections provided. We convert the CIS task into a standard information retrieval task and use both traditional IR model and neural IR model to rerank the baseline official evaluation results. We compare the results of models in two categories(four models in total), and give a summary for the solution of our work.

1 Introduction

Conversational Information Seeking is timely and important with increased adoption of a new generation of conversational ‘assistant’ systems. The focus of the TREC 2019 Conversational Assistance is on understanding of information needs in a conversational format and finding relevant responses using contextual information. Baseline retrieval run on 30 example training topics, limited training data for the 30 topics judged from the baseline retrieval run, as well as baseline retrieval results of evaluation data for 50 evaluation topics were provided as data. There are 10 queries in each topic proposed in terms of conversation, and every query has 1000 relevant documents retrieved advance in baseline retrieved results for both training data and evaluation data. This task is similar to conversational question answering or conversational information retrieving, so we see mainstream approaches in these area as follows:

1. Traditional matching methods basing on the idea of TF-IDF, such as BM25.

2. Document matching extraction based on semantic extraction.

3. Query and document's pair wise learning.

Many open source search engines provide good support for the first approach, and BERT [1] provides a strong support for the latter two methods. Based on the above research background，we implement the following four models for the TREC 2019 Conversational Assistance track:

1. For each query, we use elasticsearch to rerank the baseline retrieval run on evaluation data.

2. For each query, we split each of its pre-selected top 1000 documents in baseline retrieval run into short segments, then using BERT to get vectorized representation. For each document, we calculate each of its short segments' vectorized representation with query vectorized representation and get a match socre, and select the highest score as the document score.

3. We concatenate a query and a pre-selected document and get the vectorized representation with BERT. With those representations which have ranking label, we construct and train a pair-wise ranking model.

4. We use the representations for all tokens in BERT's last layer as the documents' and queries' word vector representation, then using Conv-knrm to calculate the final match score.

In terms of model performance, the fourth model performs better than the third model on training data set. When actually evaluating 50 topics, we select part of the results for manual analysis and find that the first model has the best performance. A detailed discussion will be covered in later sections.

2 Models

First, we use coreference resolution for each question in multi-turn conversations. Then we use the disambiguated questions as queries, and convert the Conversational Information Seeking task into a standard retrieval task. In this section, we introduce our four models designed for this task.

2.1 Elasticsearch

Elasticsearch[2] is a full-text search engine.It is because Elasticsearch contains many advantages that it becomes the most popular enterprise search engine. That is, all kinds of documents can be applied with Elasticsearch, meanwhile, with a given keyword, it can do scalable search and the results it returns usually are closer to real-time than other search engines. More importantly, with the rapid development of Internet, we have to face the phenomenon of sharp expansion of data volume, so it is important to Search in massive amounts of data and get results quickly. The distribution of Elasticsearch makes it possible to support massive, petabytes of big data searches, especially, Elasticsearch also has near real-time (second-level) performance support at massive data levels, as well as grammatical support for powerful search and aggregation analysis. All of these advantages make Elasticsearch more suitable for data analysis applications in big data scenarios.

In this task, for a variable query, we should find the most 1000 releavent document in three collections, that is, MS MARCO Passage Ranking collection, the TREC CAR paragraph collection v2.0 and the TREC Washington Post Corpus version 2, which consist of over ten million documents. First, we process all these documents into xml format with TREC-CAsT Tools. Next, we use Elasticsearch to do further search sorting based on the search results and queries given by the baseline.

Elasticsearch's scoring ideas are based primarily on bm25[3] and tfidf[4]. Lucene[5] (and thus Elasticsearch) uses the Boolean model to find matching documents, and a formula called the practical scoring function to calculate relevance. This formula borrows concepts from term frequency/inverse document frequency and the vector space model but adds more-modern features like a coordination factor, field length normalization, and term or query clause boosting. The formula of Lucene calculateing for the score is as follows:

$$\begin{aligned} score(q,d) =& queryNorm(q) \cdot coord(q,d) \\ & \cdot \sum_{t \in q} (tf(t,d) \cdot idf(t)^2 \\ & \cdot t.getBoost() \cdot norm(t,d)) \end{aligned} \tag{1}$$

$$queryNorm = \frac{1}{\sqrt{SumOfSquaredWeights}} \tag{2}$$

$$coord(q,d) = \frac{overlap}{maxoverlap} \tag{3}$$

$$tf(t,d) = \sqrt{frequency} \tag{4}$$

$$idf(t) = 1 + log\frac{numDocs}{docFreq+1} \tag{5}$$

$$\begin{aligned} sumOfSquaredWeights =& idf(t_1)^2 + idf(t_2)^2 \\ & + ... + idf(t_n)^2 \end{aligned} \tag{6}$$

$$norm(d) = \frac{1}{numTerms} \tag{7}$$

where the $score(q,d)$ is the relevance score of document d for query q. $queryNorm(q)$ is the query normalization factor. $coord(q,d)$ is the coordination factor.Where overlap is to retrieve the number of terms in the hit query, maxoverlap is the total number of terms in the query. $tf(t,d)$ is the term frequency for term t in document d. $idf(t)$ is the inverse document frequency for term t, $numDocs$ is the acount of all documents, $docFreq$ is the amount of documents that contains term t. $t.getBoost()$ is the boost that has been applied to the query,and can be seen as the weight of each term. $norm(t,d)$ is the field-length norm, combined with the index-time field-level boost, if any.$numTerms$ is terms' counts of a document.Finally, sum all of the weights for each term t in the query q for document d.

2.2 BERT Based Models

We have seen a rapid growth of pre-train neural language models recently, such as ELMo[6], OpenAI GPT[7], BERT. They promote the development of a lot of tasks in natural language understanding. We select BERT as our base model. Unlike traditional word embedding, BERT is contextual – the representation of a word is a function

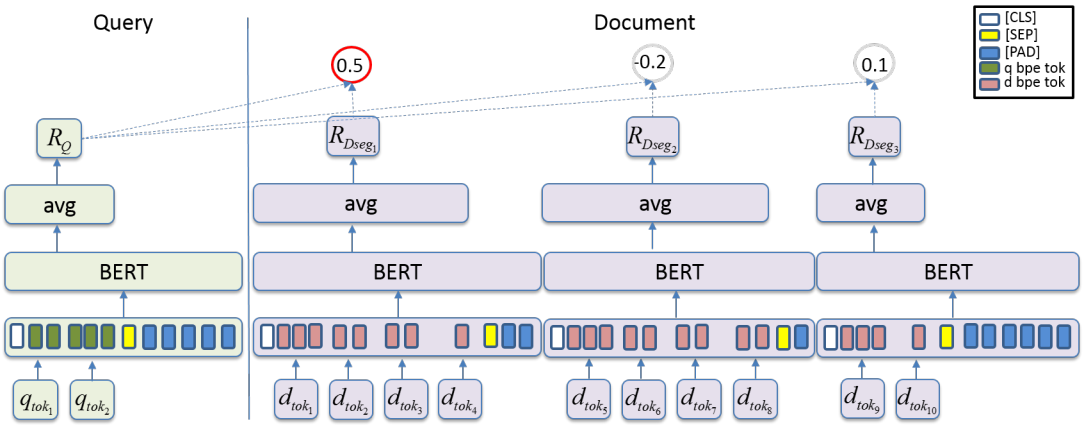

Figure 1: BERT+Sim Model

of the entire input text, taking word dependencies and sentence structures into account. BERT is pre-trained on a large corpus and thus, it can encode many language features in its contextual representation. BERT is also well suited for the retrieval task as its NSP task can help the interaction-based model judge the relationship between two pieces of text very well.

In this paper, we use the fine-tuned BERT released by [8] who augment the primitive BERT with search knowledge by continuing to train it on a large sample of Bing web search log. We use this Bing-augment BERT as our feature-extraction model and try three methods to rerank the top-k baseline results based on the contextual representation produced by this Bing-augment BERT.

2.2.1 BERT + Sim

For most documents, the length of a document is always too long compared to a query's length, and it may be inaccurate if we compute the cosine similarity between a query and a document directly. We thus split each document into many short segments. We set the length of each segment to be twice the length of a correspond query and the last segment's length may be shorter than that. We use BPE [9] to tokenize each token in each sentence and record the length of each tokenized sentence. We add token [CLS] at the start of the tokenized sentence and add [SEP] at the end of the sentence whether it is a query or a segment of a document. We set the max input length of the BERT to be a fixed number. We fill the token [PAD] if the length of the tokenized sentence is shorter than the max input length.

As shown in figure 1, for each query-document pair, we first get split tokenized segments of the document and get their tokens' embedding at the first layer of the model. Then we feed the tokens' embedding of the query and each segment into the Bing-augment BERT. We use average pooling for tokens' representations of the BERT's last layer to get the final representation of a sentence(a query or a segment) at the model's second layer.

We compute the final representation's cosine similarity between a query and each segment of a document at the last layer of the model. Then we get some similarity values for a query-document pair. For the ith segment of document, the similarity is

$$sim_i = cos(R_Q, R_Dseg\_i) \quad (8)$$

We use the max similarity value as the final similarity score for the document with respect to the query.

$$Sim = \max\{sim_i\}(i = 1, 2, .., num\_seg) \quad (9)$$

2.2.2 BERT + MLP

For this model, we don't split documents into a lot of segments, we concatenate a query and a document with a [SEP] token as shown in figure 2 instead.

First, we construct a lot of positive-negative document pairs for each query according to their different relative score for the same query. For example, two documents of relative score 2 and 0, we regard the document of score 2 as a positive sample and the document of score 0 as a negative sample.

Second, we get the BPE tokenized sentences' embedding of the two documents and padding them to the same length. For both documents, we concatenate the sequence embedding of the query

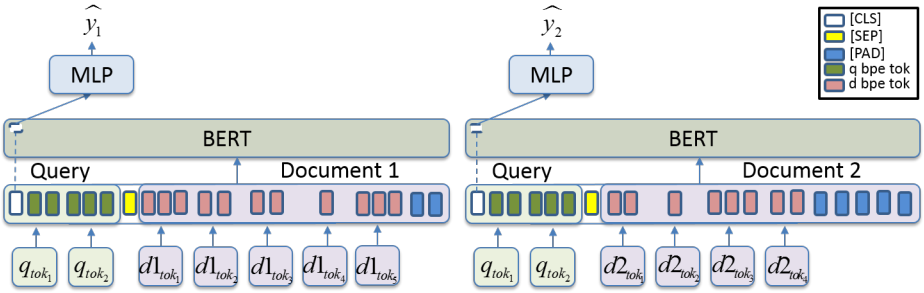

Figure 2: BERT+MLP Model

and the document with a [SEP] token's embedding. We feed the concatenated embedding for two query-document pairs into the Bing-augment BERT.

Third, we get the token [CLS]'s representation of the last BERT layer for both query-document pairs and feed two representations into a same MLP network. The MLP's output unit's dimension is one. So we get two predicted score $\hat{y_1}$ and $\hat{y_2}$ for two query-document pairs. We use the hinge loss to train our BERT + MLP model.

$$l = \sum_{q} \sum_{d^+, d^- \in D_q^{+,-}} \max(0, y(d^+) - y(d^-) - (f(q, d^+) - f(q, d^-))) \tag{10}$$

where $D_q^{+,-}$ are q' s pairwise preferences, d^+ranks higher than d^-, $y(d^+)$ and $y(d^-)$ are their true relative scores, $f(q, d^+)$ and $f(q, d^-)$ are the predicted relative scores. For the sake of brevity, we use $\hat{y_1}$ for $f(q, d^+)$ and $\hat{y_2}$ for $f(q, d^-)$ in figure 2.

2.2.3 BERT + Conv-knrm

In information retrieval task, the query and document often match at n-grams, such as phrases, concepts and entities. However, people usually treat n-grams identically to words, which greatly explode the parameter space, and suffer from data sparsity. In this paper, we use the Conv-knrm, a Convolutional Kernel-based Neural Ranking Model that models n-gram soft matches for ad-hoc search[10], which uses convolutional neural networks to represent n-grams of various lengths and soft matches them in a unified embedding space. The n-gram soft matches are then utilized by the kernel pooling and learning-to-rank layers to generate the final ranking score.

In our work, we combine the Bing-augment BERT and Conv-knrm model, in which BERT provides the query and document embedding representations and then employ Conv-knrm model on the query and document words' embeddings to get the final ranking score as showing in Figure 3. In detail, we use the query and document as input of the Bing-augment BERT respectively, and generate the output of the last layer as corresponding word embedding. While the query words' embedding and document words' embedding have been obtained, the convolutional layer applies convolution filters to compose n-grams from the text(query or document). To be detail, we will get h_{max} kinds n-grams(1, 2, ..., h_{max}), for each h-grams $h \in \{1, ..., h_{max}\}$, the CNN layer converts the text embedding into h-gram embedding G_q^h or G_d. And the same set of convolution filters is used to compose all n-grams thus the model only needs to learn the CNN weights for combining word-level embeddings, which have much fewer parameters. Then the cross-match layer matches query n-grams and document n-grams of different lengths. The unified embedding representations allow cross-matching n-grams of different lengths, and it generates h_{max}^2 translation matrices. After that, kernel-pooling which has K Gaussian kernels is applied to count the soft matches of word or n-gram pairs at K different strength levels. In this step, we get K soft-TF features for each of the h_{max}^2 translation matrices in M. Finally, a ranking score is been computed by the learning-to-rank layer which combines the soft-TF ranking features through neural network layer. For an overview, we refer to [10].

All of the BERT and Conv-knrm layers are differentiable, and standard pairwise learning-to-rank is used to train the combined model jointly.

$$l = \sum_{q} \sum_{d^+, d^- \in D_q^{+,-}} max(0, 1 - f(q, d^+) + f(q, d^-)) \tag{11}$$

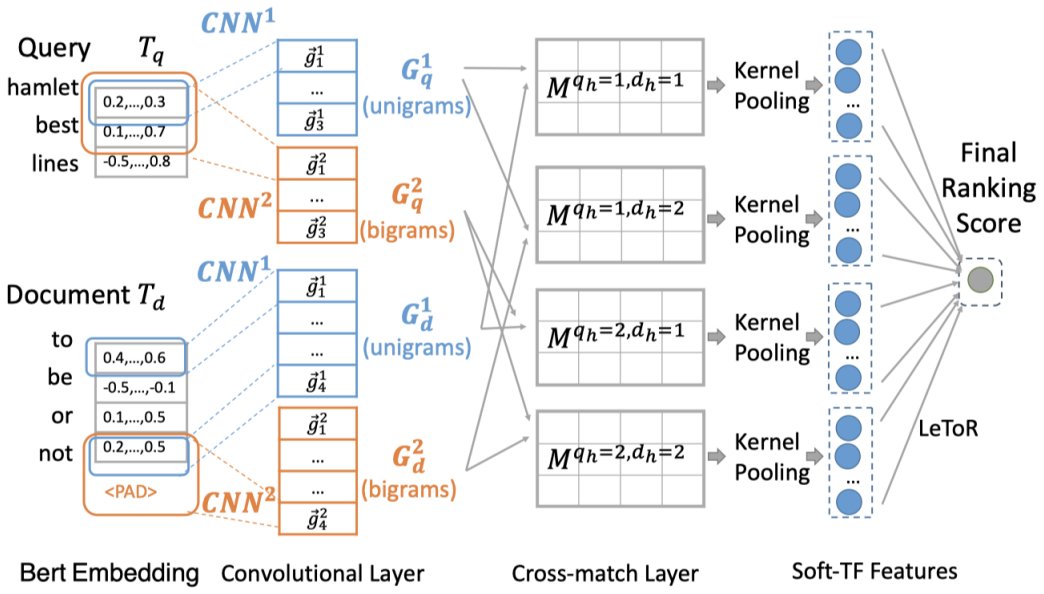

Figure 3: BERT + Conv-knrm Model

where $D_q^{+,-}$ are q' s pairwise preferences: d^+ranks higher than d^-. In our work, as show in the train data, there are three categories rank scores namely 0, 1, 2.

3 Experiments

In our experiments, we convert all sentences of the whole data set to lowercase form.

We use the training data with judgments released by Trec cast as our experiment data, there are 1930 labeled query-document pairs in total. We split the labeled data into train set, validation set and test set with the proportion of 8:1:1. For BERT+MLP model and BERT+Conv-knrm model, we use train set and validation set during the training process and test the models' ability on the test data set. Finally, we use our trained model to rerank the baseline results provided by trec cast using indri on the evaluation data containing 50 evaluation topics. For Elastic search model and BERT+Sim model, we directly rerank the baseline evaluation results provided by trec cast using the two models.

3.1 Experimental Settings

For Elastic search model, We use the default sorting algorithm of Elasticsearch, which the result is returned in reverse order according to the degree of relevance (matching degree) of the document and the query. The larger the score (_score), the higher the correlation.

The documents used for Elastic search are processed as the following examples:

```
<DOC>
    <DOCNO>CAR_000000afe1da525b3d
    b17db77f350b187441a9ed
    </DOCNO>
    <DOCHDR>

    </DOCHDR>
    <BODY>The 1913 Johannisthal A
    ir Disaster happened close to
    the air field, killing all 28
    passengers.The German astrona
    ut Rein hard Furrer died on S
    eptember 9,1995 during histor
    ic flight.
    </BODY>
</DOC>

<DOC>
    <DOCNO>MARCO_0</DOCNO>
    <DOCHDR>

    </DOCHDR>
    <HTML>
    <BODY>The presence of communi
    cation amid scientific minds
    was equally important to the
    success of the Manhattan Proj
    ect as scientific intellect w
    as. The only cloud hanging ov
```

```
    er the impressive achievement
    of the atomic researchers and
    engineers is what their succe
    ss truly meant;hundreds of th
    ousands of innocent lives obl
    iterated.
    </BODY>
    </HTML>
</DOC>

<DOC>
    <DOCNO>WAPO_b2e89334-33f9-11e
    1-825f-dabc29fd7071-1</DOCNO>
    <DOCHDR>https://www.washingto
    npost.com/sports/colleges/dan
    ny-coale-jarrett-boykin-are-a
    -perfect-1-2-punch-for-virgin
    ia-tech/2011/12/31/gIQAAaW4SP
    _story.html
    </DOCHDR>
    <HTML>
    <BODY>
    <span class="dateline">NEW OR
    LEANS ?</span> Whenever a <a
    href="http:// www.washingtonp
    ost.com/blogs/hokies-journal"
    title="www.washingtonpost.com
    ">Virginia Tech</a> offensive
    coach is asked how the most p
    rolific receiving duo in scho
    ol history came to be, inevit
    ably the first road game in 2
    008 against North Carolina co
    mes up.
    </BODY>
    </HTML>
</DOC>
```

After we getting the converted data,before using elasticsearch for search sorting, we need to create corresponding indexs for all documents.What's more,considering the large amount of data, we used the method of importing large documents specified by elasticsearch——divide large documents into smaller chunks of bulk document then import. The code are as follows:

```
#!/bin/bash

#split -l 10000 marco.json ./tmp
/macro_bulk
#BULK_FILES=./tmp/marco_bulk*
for f in ./tmp2/*; do
    curl -H "Content-Type: appli
    cation/json" -XPOST "localho
    st: 9200/test/_bulk" --data-
    binary "@$f" >> /dev/null
        temp_file=${f:16}
        part1="localhost:9200/"
        part2="/_bulk"
        #curl -H "Content-Type:a
        ppliaction/json"  -XPOST
        $part1$temp_file$part2 -
        -data-binary "@$f" >> /d
        ev/null
        echo $f >> ./import.log
done
```

Then we can set the query to search for elasticsearch. The search settings are given by the following code:

```
seach_body = {"query": {"match":
             {"id":query}}}
res = es.search(index=[all index
s], body=seach_body)
for hit in res['hits']['hits']:
    id = hit["_source"]["id"]
    body = hit["_source"]["body"]
```

All of the three BERT-based models are based on the Bing-augment BERT. It is augmented based on the primitive BERT-base-uncased model.The number of BERT layers is 12, the hidden size is 768 and the number of attention heads is 12. For BERT+Sim model, the max length of all the queries is 20 and thus we set the length of a short segment to be 40. We set the max input length of BERT to be 100. For BERT+MLP model and BERT+Conv-knrm model, we set the max input length of BERT to be 512. We use adam optimiser to train the MLP model and the Conv-knrm model with learning rate of 5e-5 , betas of (0.9,0.98) and epsilon of 1e-8. In BERT+Conv-knrm model, the max n-grams number is 3, the number of convolution filters is 128, and the number of Gaussian kernels is 11.

3.2 Experimental Results

Table 1: Performance comparisons for BERT+MLP and BERT+Conv-knrm on labeled test data set.

Model	MRR	MAP	NDCG@3	NDCG@5
BERT+MLP	0.3372	0.3060	0.2314	0.2605
BERT+CONV-KNRM	0.4368	0.4007	0.2452	0.2713

We use MRR, MAP, NDCG@3 and NDCG@5 as automatic metrics for BERT+MLP model and BERT+Conv-knrm model as those two models need training. As can be seen from table 1, We can get competitive performance using only BERT+MLP model. The Bing-augment BERT has a strong ability to model the interaction of query and document. The representations extracted by BERT help the MLP model to give a more accurate match score for query-document pairs. When we use BERT+Conv-knrm model, we have a bigger improvement in the performance of ranking retrieved documents.

Finally, we compare the performance of all the models on the evaluation data set by human evaluation. We firstly sample 10 evaluation topics with ranking results provided by four models. For each topic, we judge whether the top-10 retrieved documents for each query match the query well for all the four models. We find that the results provided by Elasticsearch has the best performance however. We finally submit the results using Elasticsearch model.

4 Results

Experimental result evaluation is generated by the assessed documents. The evaluation consists of MAP,NDCG and so on, which contains 91 metrics.And the results are as follows:

Table 2: Performance comparisons for our results and median results of all submissions.

results	MAP@5	NDCG@5	NDCG@1000
median	**0.0337**	**0.2656**	**0.3622**
elasticsearch	0.0193	0.1641	0.2574

According to the results from the official return, we made another statistics.

1. On MAP@5,We have 75 turns with a higher score than the median.

2. On NDCG@5,We have 70 turns performing better than the median.

3. On NDCG@1000,the scores of 58 turns in our method are higher than those in median.

There are 173 turns in total on the evaluation dataset, and it seems that our results is a little worse on the above three metrics. According to the model results we submitted, we can conclude that only using traditional search model solely may be not a good idea in information retrieval task because of not utilizing the ground truth. However, in this TREC, the ground truth is too few to obtain a better results for neural IR model. Thus, the lack of ground truth is a dilemma for us to train the neural IR model and get a promising results.

5 Conclusion

In this paper, we use a variety of models aiming for tackle the problem of the TREC 2019 Conversational Assistance Track, including the traditional model such as Elasticsearch and the combination BERT-based models i.e. cosine similarity, MLP and Conv-knrm. The BERT+Conv-knrm model perform better than the BERT+MLP model on the relavant documents reranking task on the training dataset. On the evaluatation dataset, we find that the Elasticsearch model has the best performance by human evaluation however. Traditional models still have strong ability in information retrieval tasks.

References

[1] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT*, pages 4171–4186, 2019.

[2] Clinton Gormley and Zachary Tong. Elasticsearch: The definitive guide. *Oreilly Media*, 2015.

[3] Robertson, Stephen, Zaragoza, Hugo, Taylor, and Michael. Simple bm25 extension to multiple weighted fields. In *Thirteenth Acm International Conference on Information & Knowledge Management*, 2004.

[4] Mark Kantrowitz, Behrang Mohit, and Vibhu O. Mittal. Stemming and its effects on TFIDF ranking. In *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, July 24-28, 2000, Athens, Greece*, pages 357–359, 2000.

[5] Joaquín Pérez-Iglesias, José R. Pérez-Agüera, Víctor Fresno, and Yuval Z. Feinstein. Integrating the probabilistic models BM25/BM25F into lucene. *CoRR*, abs/0911.5046, 2009.

[6] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[7] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2.amazonaws.com/openai-assets/researchcovers/languageunsupervised/language understanding paper.pdf*, 2018.

[8] Zhuyun Dai and Jamie Callan. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 985–988. ACM, 2019.

[9] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL*, 2016.

[10] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. Convolutional neural networks for soft-matching n-grams in ad-hoc search. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 126–134. ACM, 2018.