ICTNET at TREC 2019 Deep Learning Track

Jiangui Chen[1,2], Yinqiong Cai[1,2], Haoquan Jiang[1,2]
[1] University of Chinese Academy of Sciences, Beijing, China
[2] CAS Key Lab of Network Data Science and Technology, Institute of Computing Technology
{chenjiangui18z, caiyinqiong18s, jianghaoquan18g}@ict.ac.cn

ABSTRACT

We participated in the Deep Learning Track at TREC 2019. We built a ranking system which combines a search component based on BM25 and a semantic matching component using pretraining knowledge. Our best run achieves NDCG@10 of 0.6650, MAP of 0.2035.

1 INTRODUCTION

The Deep Learning Track aims to find the most relevant text for a given query. Normally, users send a query to the system, and the system should be able to retrieval a considerable number of documents involved with the query. After which, various methods are implemented to determine the final best answer.

Passage ranking can be seen as a text matching problem. Text matching is essential in information retrieval and natural language processing. A large majority of tasks can be formalized as a matching problem. In question answering, we need to match questions with answers. In ad-hoc retrieval, we are required to find relevant documents for a query. And in natural language inference, we expected to discover the relationship between premises and hypothesis.

Most passage ranking task has a full ranking and re-ranking subtasks. Generally, full ranking consists of two steps. The first step is to narrow the search space when concerning with efficiency. And the next step is required to dig the deep semantic information of queries and passages. Traditional methods largely rely on hand-craft features, which is inadequate and time-consuming. With the blooming of deep learning technology, a wide range of neural ranking models are applied to tackle the matching problems of queries and passages[5].

In this paper, we propose a ranking system which combines a search component based on BM25 and a semantic matching component using pretraining knowledge. Given a query Q, The search component is responsible to return relatively relevant passages. Based on the passages retrieved by search component, semantic matching component models the matching information between the query and passages.

2 SYSTEM DESCRIPTION

2.1 Overview

As Figure1 shows, given query Q, we first retrieve top 1000 relevant passages from the extended passage collection through Search Component. Then we use Semantic Matching Component to rerank the 1000 relevant passages. By comparing the final top 10 relevant passages and ground truth, we can get the evaluation metrics on query Q.

2.2 Search Component

Inspired by [9], we trained a model that predicts corresponding queries for passages and then expand the passages with those predictions. After enriching passages with extra information, we then index them. The search strategy is rough BM25 [12]. Results show that the performance (MRR) of BM25 can improve over 0.03 by expanding passages.

Concretely, we processed training data to construct query-passage pairs. The passage is the corresponding positive examples of the query. Then, we used these pairs to train a machine translation model with the help of OpenNMT [6][1], in which passage is viewed as source sequence while query is viewed as target sequence. The machine translation model uses a transformer-based encoder-decoder architecture. The encoder processes source sequence to a sequence of hidden vectors and the decoder combines hidden representations of previously generated words with source hidden vectors to predict the probability distribution of next word. The encoder and decoder are composed of six layers of transformer respectively.

After training the machine translation model, we utilized it to predict queries for each passage in the collection. In the test-time, decoding is done through beam search where multiple hypothesis target predictions are considered at each time step. Eventually we can get several predicted queries for each passage, then we concatenate its predicted queries after each passage.

Subsequently, we index these passages using Anserini [14]. Then BM25 is used to retrieve top 1000 relevant passages for each query in training and development dataset.

2.3 Semantic Matching Component

The semantic matching component aims to re-rank the documents retrieved by the search component. We have seen a rapid growth of pretrain neural models recently, such as ELMo[10], OpenAI GPT[11], BERT[2] and XLNet[15]. Models pretrained on a language modeling task achieve promising results on various natural language tasks. To leverage the ability of BERT, we introduced it as the base unit of our semantic matching component.

Given a query Q consisting of m tokens, we denotes it as $Q = q_1 q_2 \dots q_m$, where q_i is the i-th tokens of Q. Similarly, a passage P can be denotes as $P = p_1 p_2 \dots p_n$, where p_i is the i-th tokens of P and n is the number of tokens of the passage.

The semantic matching component that we integrated has two types, which are BERT and Conv-KNRM.

2.3.1 BERT. BERT was proposed by [2], which apples the bidirectional of Transformer[13] to train language model. BERT obtains new state-of-the-art results on a diversity of natural language processing tasks. The pretrained BERT model can be used to create state-of-the-art models for a wide range of task by only fine-

[1] https://github.com/OpenNMT/OpenNMT-py

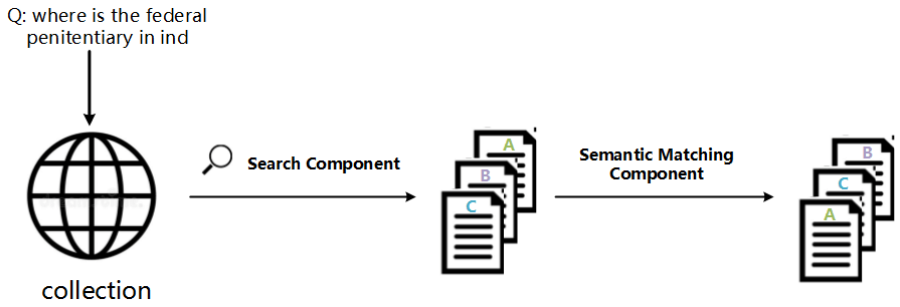

Figure 1: Overview of Ranking System

tuned with just one additional output layer, without substantial task-specific architecture modifications.

Inspired by [8], we used BERT as re-ranker to leverage the substantial ability of BERT. The query Q was marked as sentence A, and the passage P was marked as sentence B. Following the original implementation, we then fed the concatenate representation into BERT. We use BERT-base model as a regression task. It uses the representation of '[CLS]' in last layer as the matching features and combines them linearly with weight w. The ranking score from BERT includes all term pair interactions between the query and passage via the cross-match attentions of transformer. Thus it is an interaction-based matching model.

For each query and passage, BERT would eventually output a value denoting the matching score. Hence we obtain:

$$s_{ij} = BERT(Q_i, P_j) \tag{1}$$

For a query Q_i, which has k relevant passages. For each passage with respect to Q_i, we gained a list of scores, $[s_{i1}, s_{i2}, \ldots, s_{ik}]$. The top-1 passage was selected by sorting scores in descent order.

2.3.2 Conv-KNRM. Conv-KNRM, which stands for a Convolutional Kernel-based Neural Ranking Model, is a state-of-the-are neural ranking model. It was proposed by [1]. Conv-KNRM utilizes the ability of convolutional neural networks to represent n-grams of various lengths and soft matches them in a unified embedding space. Before generating the final score of each query and passage pair, kernel pooling and learning-to-rank layers are applied by feeding the n-gram soft matches signals. The architecture of Conv-KNRM is showed in Figure 2.

Query Q and passage P were first fed into embedding layer, which maps tokens of Q_i, i.e. $[q_1, q_2, \ldots, q_m]$, and tokens of P_i, i.e.$[p_1, p_2, \ldots, p_n]$ into distributed representations. Convolutional layer was followed by embedding layer to generate n-grams embedding, which takes multiple tokens into account. The key to advantages of Conv-KNRM is to use cross-matching n-grams. The query n-grams and passage n-grams of various lengths, which capture different granularity of semantic information different, were matched by cross-match layer. Translation matrices produced by cross-match layer were fed into kernel pooling layer to generate soft-TF features. In the end, learning-to-rank layer combines the highly extracted signals to generate ranking score.

Identical to Section 2.3.1, Conv-KNRM produces a list of ranking scores. After sorting scores, we obtain the highest relevant passages.

We train BERT and Conv-KNRM by using cross-entropy loss:

$$\mathcal{L} = -\sum_{j \in S_{pos}} \log(s_j) - \sum_{j \in S_{neg}} \log(1 - s_j) \tag{2}$$

where S_{pos} is the set of relevant passage with respect to query, while S_{neg} is the set of non-relevant ones.

3 EXPERIMENT

3.1 Experimental Setup

We conducted a series of experiments on TREC-Deep Learning passage ranking datasets.

Data. The passage ranking dataset is built using technology and data from Microsoft's Bing[7]. Its passage collection includes 8.8M passages extracted from 3.5M URLs. There are 1,010,916 unique real queries that were generated by sampling and anonymizing Bing usage logs. These queries are divided into three parts for train, dev and test, as Table 1 shows. It also provides qrels files which has a QID to PID mapping of when a question has had a passage marked as relevant. The average length of queries and passages is 5.9 and 57.7 respectively.

	#	#Sample
collection	total	8841823
4*query	total	1010916
	train	808731
	dev	101093
	test	101092
2*qrel	train	532761
	dev	59273

Table 1: Passage ranking dataset

Evaluate Metrics. We adopt four evaluation measures, i.e. Precision (P), Recall (R), MAP and NDCG, to evluate the performance of our proposed system.

Experimental Details.

We first used BM25 to retrieve top 1000 relevant passages for each query in train dataset. The parameter of BM25 is set to b1=0.8, k=0.6. At training time, we constructed the training set in the following way. The passages from qrels file were positive samples and

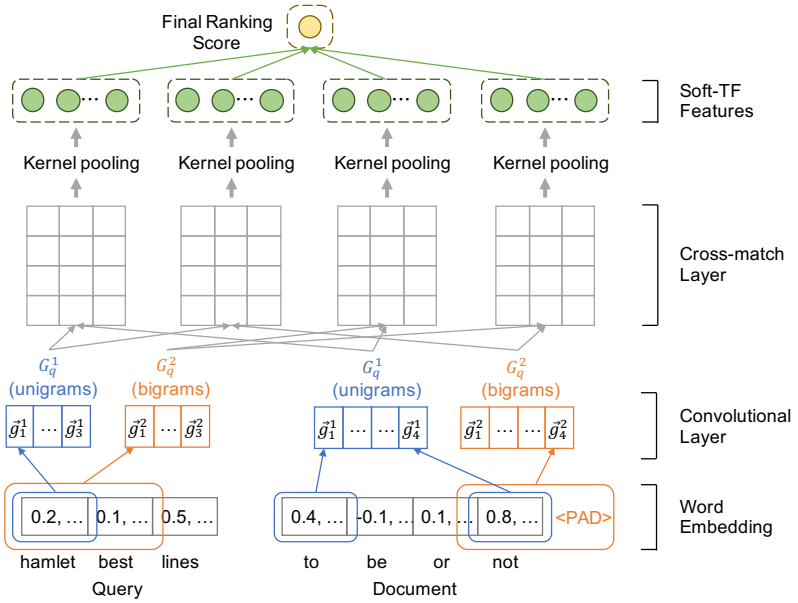

Figure 2: Architecture of Conv-KNRM

negative samples were sampled from results of BM25. Concretely, each query has 100 negative samples, which are consisted of top 50 relevant passages of BM25 and 50 passages randomly sampled from the remaining 950 relevant passages. Because each query has 1.04 positive samples on average, we duplicated each positive sample twice for making full use of negative samples. All our experiments were carried out using MatchZoo-py [3, 4][2]. With MatchZoo-py, we can easily use pairwise method for model training. A training pair is consisted of a query, a positive sample and four negative samples. For each training epoch, we resampled four negative samples for each positive sample. At prediction time, for improving performance, we only used top 50 or 100 relevant passages of BM25 for re-rank.

We conducted three experiments on required dataset, named BERT2, CKNRM_B and CKNRM_B50, respectively.

BERT2 applies BERT on the concatenated query and passage sequence. The length of query and passage is fixed as 20 and 150 respectively. BERT is fine-tuned from the pretrained BERT-BASE model released by Google. AdamW optimizer is applied to optimize the model, with 5e-5 of learning rate, 0.9 and 0.98 of betas. The warmup step is set to 6 during each epoch and batch size was 16.

CKNRM_B uses the sequence output of last layer in BERT as input. BERT was fine-tuned with Conv-KNRM end-to-end. For Conv-KNRM, max n-gram is 3, the number of Gaussian kernels is 11 and we use 128 filters in each convolution layer. The model is trained using Adam optimizer on a typical GPU, and batch size is 64. Initial learning rate is set to 1e-3. The learning rate decays with factor 0.9 every three epochs.

CKNRM_B50 uses the same model as CKNRM_B, but only utilizes top 50 relevant passages of BM25 at prediction time.

[2]https://github.com/NTMC-Community/MatchZoo-py

3.2 Result and Discussion

RUN	P@10	R@10	NDCG@10	MAP@10
BERT2	0.5581	0.2415	**0.6650**	**0.2035**
CKNRM_B	**0.5698**	**0.2437**	0.6481	0.1924
CKNRM_B50	0.5302	0.1971	0.6014	0.1404

Table 2: Evaluation results for runs based on automatic judgements

Table 2 shows the results of our submitted runs. As shown in the table, **BERT2** achieves best results although we only added a learning-to-rank layer and fine-tuned on it. This demonstrates that the effectiveness and robustness of pretraining models.

When combining pretraining language models like BERT and traditional neural ranking model Conv-KNRM, we can see a slight decrease in the evaluation results. This is because the representation produced by BERT is already highly extracted, which includes the semantic information of two original texts and the interactions between them. When feeding them into Conv-KNRM, the duplicate operations of Conv-KNRM may disturb the information that BERT has distilled previously.

4 CONCLUSION

We participated in the Deep Learning Track at TREC 2019. We proposed a ranking system which combines a search component based on BM25 and a semantic matching component using pretraining knowledge. The search component narrows down searching space for a given query, while the semantic matching component models the matching signals. Future work includes introducing external

knowledge to help modeling the matching signals between query and passages.

REFERENCES

[1] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. 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*. ACM, 126–134.

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

[3] Yixing Fan, Liang Pang, JianPeng Hou, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2017. Matchzoo: A toolkit for deep text matching. *arXiv preprint arXiv:1707.07270*.

[4] Jiafeng Guo, Yixing Fan, Xiang Ji, and Xueqi Cheng. 2019. MatchZoo: A Learning, Practicing, and Developing System for Neural Text Matching. In *Proceedings of the 42Nd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'19)*. ACM, New York, NY, USA, 1297–1300. https://doi.org/10.1145/3331184.3331403

[5] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. 2019. A deep look into neural ranking models for information retrieval. *arXiv preprint arXiv:1903.06902* (2019).

[6] Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. OpenNMT: Open-Source Toolkit for Neural Machine Translation. In *Proc. ACL*. https://doi.org/10.18653/v1/P17-4012

[7] Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A Human-Generated MAchine Reading COmprehension Dataset.

[8] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).

[9] Rodrigo Nogueira, Wei Yang, Jimmy Lin, and Kyunghyun Cho. 2019. Document Expansion by Query Prediction. *arXiv preprint arXiv:1904.08375* (2019).

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

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

[12] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[13] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*. 5998–6008.

[14] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of Lucene for information retrieval research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 1253–1256.

[15] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv preprint arXiv:1906.08237* (2019).