TU Wien @ TREC Deep Learning '19 – Simple Contextualization for Re-ranking

Sebastian Hofstätter
TU Wien
s.hofstaetter@tuwien.ac.at

Markus Zlabinger
TU Wien
markus.zlabinger@tuwien.ac.at

Allan Hanbury
TU Wien
hanbury@ifs.tuwien.ac.at

ABSTRACT

The usage of neural network models puts multiple objectives in conflict with each other: Ideally we would like to create a neural model that is effective, efficient, and interpretable at the same time. However, in most instances we have to choose which property is most important to us. We used the opportunity of the TREC 2019 Deep Learning track to evaluate the effectiveness of a balanced neural re-ranking approach. We submitted results of the TK (Transformer-Kernel) model: a neural re-ranking model for ad-hoc search using an efficient contextualization mechanism. TK employs a very small number of lightweight Transformer layers to contextualize query and document word embeddings. To score individual term interactions, we use a document-length enhanced kernel-pooling, which enables users to gain insight into the model. Our best result for the passage ranking task is: 0.420 MAP, 0.671 nDCG, 0.598 P@10 (TUW19-p3 full). Our best result for the document ranking task is: 0.271 MAP, 0.465 nDCG, 0.730 P@10 (TUW19-d3 re-ranking).

1 INTRODUCTION

Our aim in the TREC 2019 Deep Learning track was to evaluate a neural re-ranking model, which balances efficiency, effectiveness, and interpretability. We submitted runs for both the passage and document ranking tasks of the Deep Learning track. We present the TK (Transformer-Kernel) model – inspired by the success of the Transformer-based BERT model [3, 9] and the simplicity of KNRM (Kernel-based Neural Ranking Model) [14]. TK employs a small number of low-dimensional Transformer layers [13] to contextualize query and document word embeddings. TK scores the interactions of the contextualized representations with simple, yet effective soft-histograms based on the kernel-pooling technique [14]. Additionally, we enhance kernel-pooling with document length normalization (Section 2).

The main differences of TK in comparison to BERT are:

- TK's contextualization uses fewer and lower dimensional Transformer layers with less attention heads. This makes the query-time inference of TK with 2 layers 40 times faster than BERT-Base with 12 layers.
- TK contextualizes query and document sequences independently; each contextualized term is represented by a single vector (available for analysis). BERT operates on a concatenated sequence of the query and the document, entangling the representations in each layer.
- The network structure of TK makes it possible to analyze the model for interpretability and further studies. TK has an *information bottleneck* built in, through which all term information is distilled: the query and document term interactions happen in a single match matrix, containing exactly one cosine similarity value for each term pair. BERT on the other hand has a continuous stream of interactions in each layer and each attention head, making a focused analysis unfeasible.

The differences of TK to previous kernel-pooling methods are:

- KNRM uses only word embeddings, therefore a match does not have context or positional information.
- CONV-KNRM [2] uses a local-contextualization with limited positional information in the form of n-gram learning with CNNs. It cross-matches all n-grams in n^2 match matrices, reducing the analyzability.

Naturally, better efficiency and a restricted information flow through the network comes at the cost of effectiveness. This brings us to our main research question for our participation at TREC'19: *How effective is our balanced model?* To investigate this question we submitted multiple configurations of the TK model. We evaluate a GloVe embedding vs. a FastText embedding, an ensemble of multiple model instances, and a windowed-kernel-pooling for the longer document ranking.

In addition to a presentation and discussion of our TREC run results (Section 3) we showcase the analysis and interpretation capabilities of the TK model. We focus on the scenario in which a user would like to understand, for a given query, why two documents are ranked differently. We start by visualizing the word-level similarities (the interaction features) and then we report a limited number of aggregated intermediate results of important kernels (Section 4).

We publish the source code of the TK model and various neural re-ranking baselines at *github.com/sebastian-hofstaetter/transformer-kernel-ranking*. The repository contains all pre-processing and evaluation code, as well as clear and documented neural network implementations using PyTorch [10] and AllenNLP [4].

2 TK: TRANSFORMER-KERNEL MODEL

In this section, we present TK, our Transformer-Kernel neural re-ranking model. In the following, we describe how we learn contextualized term representations (Section 2.1) and how we transparently score their interactions (Section 2.2). Figure 1 gives an overview of TK's architecture.

2.1 Contextualized Term Representation

TK uses a hybrid contextualization approach. The base representations are single-vector-per-word embeddings [11]. We chose a simple word embedding structure over more complex methods – such as FastText [1] or ELMo [12] – as it offers the following benefits in practice: Word embeddings are easy to pre-train on domain specific data [6]; they require only one id per term, making the

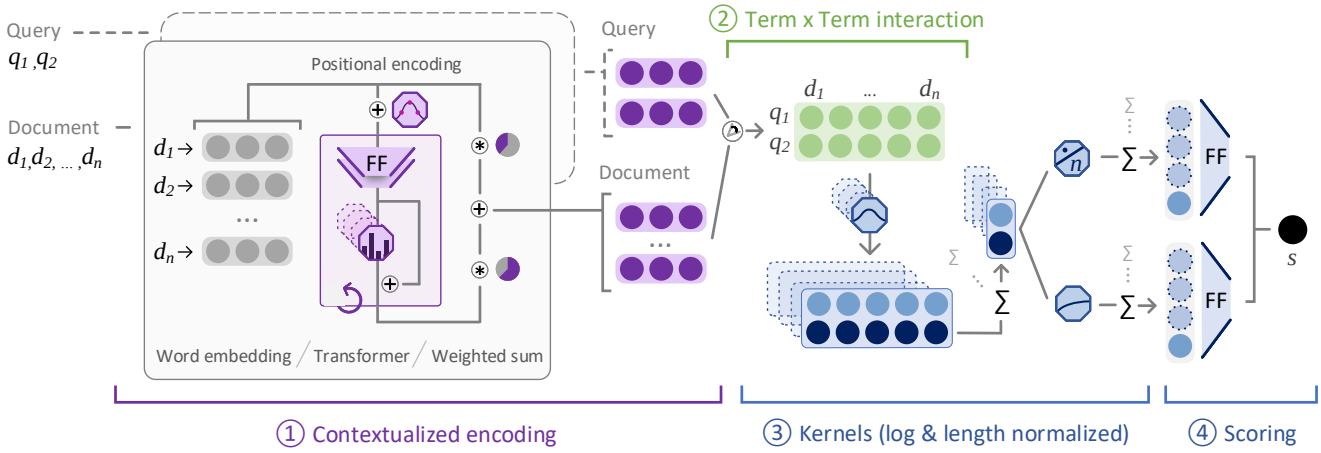

Figure 1: The TK model architecture: ① We contextualize query and document sequences individually. ② The interaction match-matrix is created with pairwise cosine similarities. ③ Each kernel creates a new feature matrix. Then, the document dimension is summed and we normalize each query-term feature by logarithm and document length. ④ We combine log- and length-normalized scores to form the final result score.

index consume less disk space, once prepared for re-ranking; most importantly, at query time, their selection is a fast memory lookup.

In the contextualization phase of the TK model, we process query $q_{1:m}$ and document sequences $d_{1:n}$ separately, however the learned parameters are shared. The input consists of two sequences of query and document ids. We employ the lookup based word embedding to select non-contextualized representations for each term. The hybrid-contextualized representation $\hat{t}_i$ of a term with word embedding t_i over its whole input sequence $t_{1:n}$ is defined as:

$$\hat{t}_i = t_i * \alpha + \text{context}(t_{1:n})_i * (1 - \alpha) \tag{1}$$

We regulate the influence of the contextualization by the end-to-end learned α parameter. This allows the model to decide the intensity of the contextualization. We calculate the context$(t_{1:n})$ with a set of Transformer layers [13]. First, the input sequence is fused with a positional encoding to form $p_{1:n}$, followed by a set of l Transformer layers:

$$\text{Transformer}_1(p_{1:n}) = \text{MultiHead}(\text{FF}(p_{1:n})) + \text{FF}(p_{1:n}) \tag{2}$$

Here, FF is a two-layer fully connected feed-forward layer including a non-linear activation function. The *MultiHead* module projects the input sequence (stored as a matrix) to query, key, and value inputs of the scaled dot-product attention for each attention head. Then the results of the attention heads are concatenated and projected to the output:

$$\text{MultiHead}(p_{1:n}) = \text{Concat}(head_1, ..., head_h)W^O$$
$$\text{where } head_i = \text{softmax}\left(\frac{(p_{1:n}W_i^Q)(p_{1:n}W_i^K)^T}{\sqrt{d_k}}\right)(p_{1:n}W_i^V) \tag{3}$$

We select Transformers for contextualization, because their positional encoding and sequence wide self-attention allows for local and global contextualization at the same time. This makes TK more powerful than previous local-only contextualization methods used in CONV-KNRM [2] and CO-PACRR [7].

2.2 Interaction Scoring

After the contextualization, we match the query sequence $\hat{q}_{1:m}$ and document sequence $\hat{d}_{1:n}$ together in a single match-matrix $M \in \mathbb{R}^{q_{\text{len}} \times d_{\text{len}}}$ with pairwise cosine similarity as interaction extractor:

$$M_{i,j} = \cos(\hat{q}_i, \hat{d}_j) \tag{4}$$

Then, we transform each entry in M with a set of k RBF-kernels [14]. Each kernel focuses on a specific similarity range with center μ_k. The size of all ranges is guided by σ. In contrast to Xiong et al. [14] we do not employ an exact match kernel – as contextualized representation are not producing exact matches. Each kernel results in a matrix $K \in \mathbb{R}^{q_{\text{len}} \times d_{\text{len}}}$:

$$K_{i,j}^k = \exp\left(-\frac{(M_{ij} - \mu_k)^2}{2\sigma^2}\right) \tag{5}$$

Now, we process each kernel matrix in parallel, and we begin by summing the document dimension j for each query term and kernel:

$$K_i^k = \sum_j K_{i,j}^k \tag{6}$$

At this point – as shown in Figure 1 – the model flow splits into two paths: log normalization and length normalization. The log normalization applies a logarithm with base b to each query term before summing them up:

$$s_{\text{log}}^k = \sum_i \log_b\left(K_i^k\right) \tag{7}$$

We enhance the pooling process with document length normalization. We dampen the magnitude of each query term signal by the document length:

$$s_{\text{len}}^k = \sum_i \frac{K_i^k}{d_{\text{len}}} \tag{8}$$

Now, each kernel is represented by a single scalar, which is weighted with a simple linear layer to produce a scalar, for both the log-normalized and length normalized kernels:

$$s_{\log} = s^k_{\log} W_1 \qquad s_{\text{len}} = s^k_{\text{len}} W_2 \tag{9}$$

Finally, we compute the final score of the query-document pair as a weighted sum of the log-normalized and the length-normalized scores:

$$s = s_{\log} * \beta + s_{\text{len}} * \gamma \tag{10}$$

We employ kernel-pooling, because it makes inspecting temporary scoring results more feasible compared to pattern based scoring methods. Each kernel is applied to the full document. The row-wise and the column-wise summing of the match-matrix allow to inspect individual matches independent from each other.

3 TREC DEEP LEARNING TRACK

Now, we describe the details of our experimentation pipeline (Section 3.1), configuration settings of our different runs (Section 3.2) and the results of the TREC annotations (Section 3.3).

3.1 Experiment Setup

Our experimentation pipeline has two parts: 1) A first stage index and retrieval and 2) neural re-ranking model training and inference. For the first part we use the Anserini toolkit [15] to compute the initial ranking lists, which we use to generate training and evaluation inputs for the neural models (only for the *full* task). As basis for our neural model library we use PyTorch [10] and AllenNLP [4]. We tokenize the text with the fast BlingFire library[1]. We train all neural models with a pairwise hinge loss. We use pre-trained GloVe [11] word embeddings with 300 dimensions[2].

We cap the query length at 30 tokens and the document length at 200 tokens. For MSMARCO-Passage this only removes a modest amount of outliers, however, for the MSMARCO-Document collection a majority of documents is longer than 200 tokens. Increasing the cap to fully include most documents would render all evaluated neural IR models less effective. Only the TUW19-d3 model uses a larger cutoff of 800. We use the Adam [8] optimizer with a learning rate of 10^{-4} for word embeddings and contextualization layers, 10^{-3} for all other network layers. We employ early stopping, based on the best MRR@10 value of the validation set. We use a training batch size of 64. We use a vocabulary of all terms with a minimum collection occurrence of 5. Regarding model-specific parameters, for the Transformer layers in TK we use 2 layers, each with 16 attention heads with size 32 and a feed-forward dimension of 100. For log-normalization in TK we use a base of 2. For kernel-pooling we set the number of kernels to 11 with the mean values of the Gaussian kernels varying from -1 to $+1$ and standard deviation of 0.1 for all kernels.

3.2 Run overview

Our runs are described in Table 1. We used the same model instance for both the evaluation of the *re-ranking* task and the *full ranking* task. For the *full* task we generated initial rankings with Anserini using BM25 and utilized the validation sets to tune the re-ranking depth. For the *re-ranking* task we used the entire provided initial ranking list.

Table 1: Summary of our submitted TK runs

Run	Description
	Passages
TUW19-p1	Using GloVe pre-trained (with min. 5 occurrence threshold), best validation run of multiple inits
TUW19-p2	Using FastText vectors instead of a simple word embedding
TUW19-p3	Ensemble of multiple TUW19-p1 configurations
	Documents
TUW19-d1	Using GloVe and document training data
TUW19-d2	Using GloVe and passage training data
TUW19-d3	Using FasText embeddings & windowed-kernel-pooling of different sizes, final score based on sorted window scores

The *TUW19-d3* model is the only submitted run diverging from the TK model description in Section 2. It caps documents at 800 tokens and contextualizes the sequences in one block. Then, after the kernel-transformation of the cosine interactions is computed, we apply multiple pooling windows of different window sizes to the soft-histogram features. After that, we sort the window-scores and weigh the position independent sorted scores to form the final score.

3.3 Results

Now, we present our results for our validation set (sparsely labeled MSMARCO-DEV set) and the TREC judgements in Table 2. Additionally, we highlight qualitative examples of the best and worst queries for two runs in Tables 3 and 4.

3.3.1 Passage Task. For the passage task our results show that different configurations of TK have similar results. Especially, the difference between a GloVe embedding with a minimum threshold of 5 (p1) and a FastText embedding without out-of-vocabulary (OOV) terms (p2) is marginal. We assume this is due to the ability of the contextualization to overcome OOV and infrequent terms, which have been shown to negatively impact simpler neural models [5]. Ensembling a model (p3), does provide some benefit, however, the difference is stronger in the loosely judged DEV set and smaller in the thoroughly judged TREC annotations.

In addition to the evaluation metrics, we selected the best and worst performing queries from the TREC'19 set and show them in Table 3. Due to the small amount of queries evaluated, there is no clear distinction in the types of information needs that perform better or worse.

3.3.2 Document Task. For the document ranking the different configurations and network structures of TK seem very similar when looking at the DEV set in Table 2. However, the TREC annotations reveal large differences between the full and re-Ranking task as well as the submitted configurations.

[1] https://github.com/microsoft/BlingFire
[2] 42B CommonCrawl lower-cased: *https://nlp.stanford.edu/projects/glove/*

Table 2: Evaluation results of our runs for the passage and document tasks.

	Run	MSMARCO-DEV			TREC2019-Full			TREC2019-ReRank		
		MAP	nDCG	MRR@10	MAP	nDCG	P@10	MAP	nDCG	P@10
Passage	**TUW19-p1**	0.314	0.366	0.307	0.413	0.667	0.574	0.407	0.640	0.570
	TUW19-p2	0.316	0.369	0.310	0.416	0.671	0.577	0.396	0.636	0.565
	TUW19-p3	**0.333**	**0.386**	**0.328**	**0.420**	**0.671**	**0.598**	**0.411**	**0.641**	**0.577**
Doc.	**TUW19-d1**	0.311	0.366	0.306	0.165	0.314	0.626	0.252	0.445	0.688
	TUW19-d2	0.312	0.365	0.303	**0.205**	**0.382**	**0.633**	0.239	0.445	0.681
	TUW19-d3	**0.314**	**0.369**	**0.309**	0.184	0.333	0.626	**0.271**	**0.465**	**0.730**

Table 3: Best & worst queries for TUW19-p3 full

Best		
Id	**AP**	**Query Text**
146187	0.851	difference between a mcdouble and a double cheeseburger
156493	0.8225	do goldfish grow
168216	0.9495	does legionella pneumophila cause pneumonia
359349	0.788	how to find the midsegment of a trapezoid
855410	1	what is theraderm used for
Worst		
Id	**AP**	**Query Text**
1063750	0.015	why did the us volunterilay enter ww1
1110199	0.0911	what is wifi vs bluetooth
1112341	0.1056	what is the daily life of thai people
1113437	0.095	what is physical description of spruce
19335	0	anthropological definition of environment
207786	0.0919	how are some sharks warm blooded
489204	0.0583	right pelvic pain causes

Table 4: Best & worst queries for TUW19-d3 re-rank

Best		
Id	**AP**	**Query Text**
287683	0.7917	how many liberty ships were built in brunswick
405717	0.4675	is cdg airport in main paris
855410	1	what is theraderm used for
962179	0.7067	when was the salvation army founded
Worst		
Id	**AP**	**Query Text**
1037798	0.0076	who is robert gray
1063750	0.0389	why did the us volunterilay enter ww1
1106007	0.0414	define visceral?
1112341	0.0277	what is the daily life of thai people
443396	0.0316	lps laws definition
451602	0.0276	medicare's definition of mechanical ventilation
47923	0.0625	axon terminals or synaptic knob definition
489204	0.0736	right pelvic pain causes

We assume that a major factor for the differences between *full* and *re-ranking* tasks is that for the *full* task we tuned the re-ranking depth on the MRR@10 score of the validation set as proposed by Hofstätter et al. [5]. For the passage models we found the maximum evaluated depth (1000 documents per query) to be the best, but for the document task the tuned threshold (on the MSMARCO-DEV set) is much lower than the 100 documents of the full ranking task. The re-ranking depths are 29 for *TUW19-d1*, 60 for *TUW19-d2* and 31 for *TUW19-d3*. For the sparsely judged DEV set this brings improvements, however for the thoroughly judged TREC queries we decrease the effectiveness substantially between 31 and 100 re-ranked documents. This shows the importance of evaluating thoroughly judged queries and the need to revisit the threshold parameter for the new dataset.

The best performing TK model (d3) is document-specific with windowed kernel-pooling and tuned re-ranking depth. This result shows that the pure passage ranking TK model (d1,d2) is unsuited for documents and we explored a first strategy on handling longer text, still we believe there is more potential for special network architectures for documents in the future.

Following the passage result, we also highlight the best and worst performing queries on the document task for the d3 run and the re-ranking task in Table 4.

4 INTERPRETABILITY

We now highlight the interpretation capabilities of the TK model with a qualitative example from the MSMARCO-Passage-DEV set. We focus on the following scenario: a user would like to know why the neural model replaced the first result (a non-relevant document) of the first stage ranking with an actual relevant document. For this, we offer a side-by-side comparison view of two documents.

Figure 2 shows the comparison of two documents for the query *"androgen receptor define"*. On the left side is a document judged as relevant, which is placed on the first position by TK. On the right side is the formerly first document (as determined by BM25), which is not the correct answer and only partially relevant to the query – TK moved it to a lower position.

We show each document with its full-text and a selection of temporary results of TK. We aim to identify and highlight the differences that result in different ranking scores. We color words according to their closest affiliation with a kernel. An important fact to consider is the soft-matching nature of the kernels: A term is counted in more than one kernel at a time. For example, this explains the difference in kernel $\mu = 1$, even though no word is more closely associated with that kernel and therefore we omitted a color.

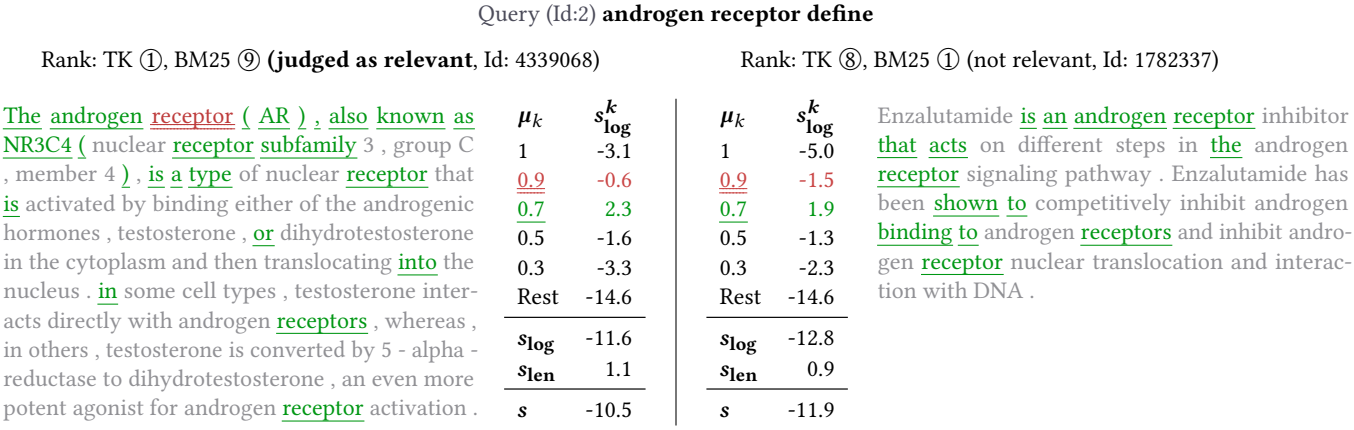

Figure 2: TK's scoring results of two MSMARCO-Passage documents: We highlight two of the most distinct kernels – both indicating close similarities (0.9 & 0.7). In the text words are colored and underlined if they are closest to the center of the highlighted kernel. Individual kernel results (model weights included) are displayed in the middle for each document.

From the highlighted kernel scores ($s^k_{\log}$) it is apparent that the left document has more stronger matches than the right one, leading to higher scores. If we look at the corresponding colored words we observe that the sentence containing the definition in the left is most relevant to the query: *The androgen receptor (AR) , also known as NR3C4 (nuclear receptor subfamily*. Even though TK does not contain a mechanism for strictly categorizing a region as relevant, it does so indirectly by strongly matching most terms in this region. Of particular interest to us is the fact that the contextualization of TK learns to match the query term *"define"* with words and phrases that make up a definition: *"also known as"*, *"subfamily"*, *"is a type"* as well as the parentheses. This exceeds simple synonym mapping, suggesting once more the importance of training contextualized and relevance specific encoding models.

This analysis demonstrates the potential for future work on keyword based search. When a collection is not queried with natural language questions, but only keywords, one could expand such keyword queries with terms like *"definition"* or *"meaning"* both during training and inference of neural models, to promote documents closer related to the core of the information need. We are aware that our approach does not enable full interpretability. We do not look deeper than the pairwise similarity values, and we currently cannot explain why certain words are similar and why some are not. Additionally, an interactive version as part of a search result page would allow more flexibility to explore different kernels and query terms. Nevertheless, we view this approach as a first step to open up the black-box of the neural re-ranking model.

5 CONCLUSION

Our aim in the TREC 2019 Deep Learning track was to evaluate a neural re-ranking model, which balances efficiency, effectiveness, and interpretability. We submitted results of our TK (Transformer-Kernel) model: a neural re-ranking model for ad-hoc search using an efficient contextualization mechanism. For the passage task our results show that different configurations of TK lead to similar results. Ensembling a model does provide some benefit, however the difference is stronger in the loosely judged DEV set and smaller in the thoroughly judged TREC annotations. For the document ranking the different configurations and network structures of TK seem very similar when looking at the DEV set. The TREC annotations however reveal large differences between the full and re-ranking task and the submitted configurations. The best performing TK model is document-length-specific with windowed kernel-pooling and tuned re-ranking depth. This shows the potential for specialized architectures for neural document re-ranking.

REFERENCES

[1] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching Word Vectors with Subword Information. *Tr. of the ACL* 5 (2017).
[2] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search. In *Proc. of WSDM*.
[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proc. of NAACL*.
[4] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. AllenNLP: A Deep Semantic Natural Language Processing Platform. arXiv:arXiv:1803.07640
[5] Sebastian Hofstätter, Navid Rekabsaz, Carsten Eickhoff, and Allan Hanbury. 2019. On the Effect of Low-Frequency Terms on Neural-IR Models. In *Proc. of SIGIR*.
[6] Sebastian Hofstätter, Navid Rekabsaz, Mihai Lupu, Carsten Eickhoff, and Allan Hanbury. 2019. Enriching Word Embeddings for Patent Retrieval with Global Context. In *Proc. of ECIR*.
[7] Kai Hui, Andrew Yates, Klaus Berberich, and Gerard De Melo. 2018. Co-PACRR: A context-aware neural IR model for ad-hoc retrieval. In *Proc. of WSDM*.
[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).
[9] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).
[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *Proc. of NIPS-W*.
[11] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proc of EMNLP*.
[12] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proc. of NAACL*.
[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 *Proc. of NIPS*.
[14] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. In *Proc. of SIGIR*.
[15] Peilin Yang, Hui Fang, and Jimmy Lin. 2017. Anserini: Enabling the use of Lucene for information retrieval research. In *Proc. of SIGIR*.