Evaluating Transformer-Kernel Models at TREC Deep Learning 2020

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

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

ABSTRACT

We tested multiple hypotheses using the Transformer-Kernel neural ranking pattern. The TK model family sits between BERT and previous ranking model in terms of the efficiency-effectiveness trade-off, faster than BERT albeit less effective.

In the passage re-ranking task we tested the effectiveness of contextualized stopwords, introduced with TK-Sparse and find that removing 19% of terms after contextualization even slightly increases the model's effectiveness. In the document re-ranking task we tested if a long-text TKL model is better with 2,000 or 4,000 document tokens and find that our 2,000 token instance outperforms the other.

Our results confirm the path for new storage saving methods for interpretable ranking models, and give an interesting insight into the questions of how many tokens of a document we need to read for a relevance assessment.

1 INTRODUCTION

In the 2020 installment of the TREC Deep Learning track, our group tested multiple hypotheses centered around the Transformer-Kernel neural ranking pattern. We introduced the TK model last year [5], as a very efficient alternative to BERT based ranking approaches [6, 7]. The TK pattern fills the gap in the efficiency-effectiveness trade-off map between BERT and non-Transformer based text ranking models [2]. Our shallow Transformers are faster than BERT, albeit less effective. In Table 1 we give a summary of our submitted runs corresponding to our two research questions.

Last year at TREC-DL'19 [1] we tested the general concept of our TK family, to understand whether the proposed concept works in general. For this year's TREC-DL passage task we tested the effectiveness of contextualized stopwords, introduced with TK-Sparse [3], as a way to reduce the number of terms saved when pre-computing document representations against a full TK model. We studied the question:

RQ1 How do space-saving contextualized stopwords effect the re-ranking quality of TK?

We found that the TK-Sparse run is slightly more effective than the plain TK run, suggesting that not only do space saving methods improve efficiency, our contextualized stopwords apparently reduce noise and improve the effectiveness of the TK model.

In the document task we tested our hypothesis of how much of a document we need to read to effectively train and run a neural ranking model. For this purpose we submitted two runs of our TKL model (Transformer-Kernel for Long documents) [4] one receiving the first 2,000 and the other the first 4,000 tokens of a document and studied:

RQ2 How effective is the 2K vs. 4K document length TKL model?

Table 1: Summary of our submitted TREC'20 runs

Run	Description
	Passages
TUW-TK-2Layer	A plain TK [5] model with GloVe embeddings and pre-trained Transformer layers on the collection
TUW-TK-Sparse	TK-Sparse [3] with 19% sparsity of stored passage representations
	Documents
TUW-TKL-2k	A TKL model [4] using 2,000 document tokens
TUW-TKL-4k	A TKL model [4] using 4,000 document tokens

Here we found that the 2K model outperforms the 4K model in the TREC 2020 evaluation. This an interesting, as we observed reverse results using the TREC 2019 query set. For efficiency, of course computing fewer tokens is beneficial, so these TREC 2020 results are encouraging for models using not all available tokens to locate relevance.

We used our PyTorch [8] implementations available at: *github.com/sebastian-hofstaetter/transformer-kernel-ranking*

2 BACKGROUND

In the following we give a quick overview of the methodology, for more details we refer to the respective papers.

2.1 TK

Our *TUW-TK-2Layer* run is an instance of the TK model. The Transformer-Kernel (TK) model [5] is not based on BERT pre-training, but rather uses shallow Transformers. TK independently contextualizes query $q_{1:m}$ and passage $p_{1:n}$ based on pre-trained word embeddings, where the intensity of the contextualization (Transformers as TF) is set by a gate α:

$$\begin{aligned} \hat{q}_i &= q_i * \alpha + \text{TF}(q_{1:m})_i * (1 - \alpha) \\ \hat{p}_i &= p_i * \alpha + \text{TF}(p_{1:n})_i * (1 - \alpha) \end{aligned} \tag{1}$$

The sequences $\hat{q}_{1:m}$ and $\hat{p}_{1:n}$ interact in a match-matrix with a cosine similarity per term pair and each similarity is activated by a set of Gaussian kernels [9]:

$$K^k_{i,j} = \exp\left(-\frac{\left(\cos(\hat{q}_i, \hat{p_j}) - \mu_k\right)^2}{2\sigma^2}\right) \tag{2}$$

Kernel-pooling is a soft-histogram, which counts the number of occurrences of similarity ranges. Each kernel k focuses on a fixed range with center μ_k and width of σ.

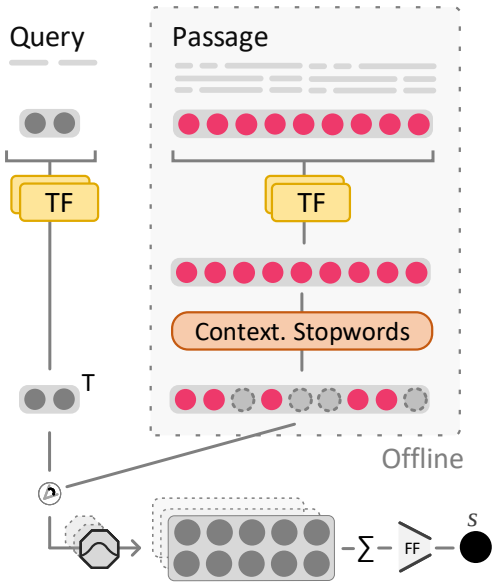

Figure 1: TK-Sparse architecture diagram: The offline pre-computed document representations are filtered through the sparse stopword component before being saved and used in the aggregation.

These kernel activations are then summed, first by the passage term dimension j, log-activated, and then the query dimension is summed, resulting in a single score per kernel. The final score is calculated by a weighted sum using W_s:

$$\text{TK}(q_{1:m}, p_{1:n}) = \left(\sum_{i=1}^{m} \log \left(\sum_{j=1}^{n} K_{i,j}^{k} \right) \right) * W_s \tag{3}$$

2.2 TK-Sparse

The adaption of contextualized stopwords used in the *TUW-TK-Sparse* run augment the TK model with a sparsity component, that filters out passage terms after the contextualization. We give an overview of this procedure in Figure 1. Every vector $\hat{d}_j$ is transformed by two feed forward layers, followed by a ReLU activation, to compute the stopword removal gate r_j:

$$r_j = \text{ReLU}(\text{FF}(\hat{d}_j) \tag{4}$$

This removal gate is applied to the kernel activations, to filter out activations that become 0 because of the gate r_j.

$$K_{i,j}^{k} = K_{i,j}^{k} * r_j \tag{5}$$

Now, we only need to store non-zero passage terms for the query-dependent element-wise kernel activation and aggregation.

The sparsity is trained with an augmented loss and L1-norm regularization of r, forcing r to become small and with the ReLu activation become zero at as many positions as possible without compromising the effectiveness of the model. For the exact procedure and training adaptions we refer to Hofstätter et al. [3].

Table 2: Official TREC'20 passage re-ranking results.

Run	nDCG@10	MRR@1K	MAP@1K
TUW-TK-2Layer	0.6539	0.7654	**0.4179**
TUW-TK-Sparse	**0.6610**	**0.7970**	0.4164

Table 3: Official TREC'20 document re-ranking results.

Run	nDCG@10	MRR@100	MAP@100
TUW-TKL-2k	**0.5852**	**0.9296**	**0.3810**
TUW-TKL-4k	0.5749	0.9185	0.3749

2.3 TKL

Our document runs *TUW-TKL-2k* and *TUW-TKL-4k* are based on TKL. The TKL model utilizes the same building blocks as TK, namely shallow transformers and an enhanced kernel pooling, but applies them in sliding windows to form local-attention regions and then selects the most relevant regions to form the document score. The exact definitions are detailed in Hofstätter et al. [4].

3 RESULTS

To answer our **RQ1** *How do space-saving contextualized stopwords effect the re-ranking quality of TK?* we show the passage re-ranking task results in Table 2. We see that TK-Sparse outperforms TK on nDCG and MRR, however on MAP TK shows better results. The main take-away here is that TK-Sparse is at least as effective as TK, showing the applicability of our contextualized stopwords to save pre-computation storage. We view this technique as an interesting path forward in combination with other vector compression techniques.

Our document re-ranking task results concerning **RQ2** *How effective is the 2K vs. 4K document length TKL model?* are shown in Table 3. The results are unexpected, yet interesting, as the previous TREC-DL'19 results for those two model instances showed the reverse outcome in Hofstätter et al. [4]. The 2K model outperforms the 4K instance on all metrics. This shows that apparently 2,000 document tokens are enough in the re-ranking scenario. For next year's TREC we plan to test this hypothesis also in the retrieval stage.

4 CONCLUSION

We submitted 4 runs to the TREC-DL'20 campaign, and studied 2 research questions concerning the effectiveness of the TK model family. We find, that our TK-Sparse model does not negatively impact the effectiveness when reducing the passage terms with contextualized stopwords. Furthermore, we observe that 2,000 document tokens seem to be enough to re-rank documents with TKL.

REFERENCES

[1] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2019. Overview of the TREC 2019 deep learning track. In *TREC*.
[2] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. arXiv:cs.IR/2010.02666

[3] Sebastian Hofstätter, Aldo Lipani, Markus Zlabinger, and Allan Hanbury. 2020. Learning to Re-Rank with Contextualized Stopwords. In *Proc. of CIKM*.
[4] Sebastian Hofstätter, Hamed Zamani, Bhaskar Mitra, Nick Craswell, and Allan Hanbury. 2020. Local Self-Attention over Long Text for Efficient Document Retrieval. In *Proc. of SIGIR*.
[5] Sebastian Hofstätter, Markus Zlabinger, and Allan Hanbury. 2020. Interpretable & Time-Budget-Constrained Contextualization for Re-Ranking. In *Proc. of ECAI*.
[6] Sean MacAvaney, Andrew Yates, Arman Cohan, and Nazli Goharian. 2019. CEDR: Contextualized Embeddings for Document Ranking. In *Proc. of SIGIR*.
[7] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085* (2019).
[8] 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*.
[9] 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*.