Melhoria da Função de Agregação do DuoT5

Leonardo Bernardi de Avila, Pedro Gabriel Gengo Lourenço
Mentor: Rodrigo Nogueira

Dezembro 2021

Resumo

O DuoT5 é um dos melhores reranqueadores que existe, porém, não funciona bem quando tem que reranquear muitos textos. O problema está possivelmente em sua função de agregação, onde, dado dois textos i e j, sendo i um texto relevante e j um texto não relevante mas sobre um assunto parecido, a chance de j ter um score maior que i ao acaso aumenta com o tamanho da lista de textos a serem reranqueados. O objetivo desse trabalho é explorar diferentes funções de agregação a fim de reduzir tal efeito ao se aumentar o número de textos. Conseguimos obter melhores resultados para NDCG@10 e NDCG@20 dos que o previamente reportado e, além disso, encontramos funções que apresentaram característica monotônica a medida que se aumenta o número de textos candidatos.

1 Introdução

O objetivo de um modelo de ranqueamento em uma tarefa de busca ad-hoc consiste em: dado um corpus de textos $C = \{d_1, d_2, ..., d_i\}$, retornar uma lista ordenada de k textos $\{d_1, d_2, \ldots, d_k\}$ em resposta a uma necessidade de informação q que um usuário tem de forma a maximizar alguma métrica de qualidade como nDCG ou MRR [4].

Para esse tipo de tarefa, já foi apresentado que uma arquitetura "retrieve-then-rerank"pode ser muito efetiva [1]. De acordo com [7], o primeiro estágio ("retrieval") dessa arquitetura pode ser realizado por um método tradicional de busca como o BM25 [9]. Já o segundo estágio ("reranking") tipicamente é executado por transformers pré-treinados, como o BERT [3], ou uma de suas variantes que tiveram um fine-tuning realizado com pares de query e textos relevantes [6].

Em [7], essa abordagem foi refinada e uma nova arquitetura foi apresentada. Ao invés de se basear em um segundo estágio executado pelo BERT, os autores propuseram a utilização de uma arquitetura multi-estágio contendo transformers sequence-to-sequence pré-treinados, no caso, baseados no T5 [8]. Essa nova arquitetura foi chamada de Expando-Mono-Duo e cada um de seus estágios $\{H_0 : BM25, H_1 : monoT5, H_2 : duoT5\}$ pode ser visualizado na Figura 1.

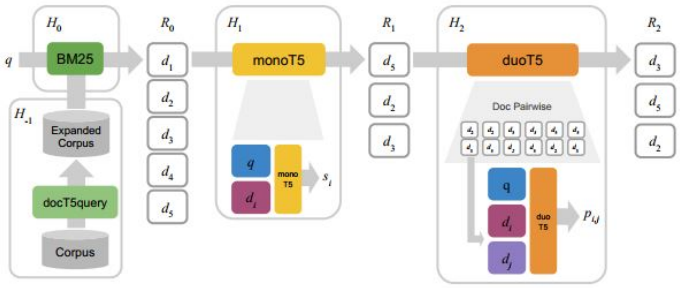

Figura 1: Ilustração da Arquitetura Multi-Estágio do Expando-Mono-Duo.[7]

Antes do primeiro estágio dessa arquitetura, temos uma etapa de expansão inicial (H_{-1}) que é referenciada no nome da arquitetura pelo termo "expando". Essa etapa tem por objetivo enriquecer a representação de palavras dos textos do corpus para que o primeiro estágio da arquitetura de ranqueamento consiga alcançar um melhor desempenho, visto que a função de ranqueamento utilizada no primeiro estágio se baseia na ocorrência de palavras no texto. Isso acontece através da utilização de um modelo doc2query nos textos do corpus onde, dado uma passagem de entrada, o modelo retorna queries que a passagem seria capaz de responder. As queries são então inseridas no texto fonte, criando uma versão expandida do mesmo. As queries preditas podem ter palavras que não estavam presentes na passagem original, então a versão expandida da passagem teria uma maior representatividade de palavras. O doc2query utilizado na arquitetura foi um docT5query [5].

No primeiro estágio (H_0), a arquitetura recebe a query q de entrada do usuário e produz como resultado os top k_0 candidatos (conjunto R_0). Na implementação do "Expando-Mono-Duo", essa etapa trata cada query como uma bag of words para que as passagens do corpus possam ser ranqueadas através do BM25 [7]. Ainda no artigo, é comentado que as queries podem também ser expandidas através da técnica de pseudo-relevance feedback, que não será abordada aqui.

No segundo estágio (H_1), os textos retornados pelo BM25 são reranqueados pelo monoT5, um reranqueador pointwise. Esse modelo estima um score s_i que quantifica o quão relevante um texto candidato, retornado pelo estágio anterior, é para uma query q [7]. Nessa etapa, espera-se que as métricas de avaliação alcançadas pelos textos reranqueados pelo monoT5 alcancem melhores resultados que o estágio anterior. Apesar disso, como trata-se de um modelo sequence-to-sequence baseado em transformer, esse processo pode se tornar muito custoso conforme aumenta a quantidade de candidatos, sendo assim, o primeiro estágio tem uma função importante de fornecer bons candidatos ao monoT5 para que então eles possam ser reranqueados. Aqui é importante ressaltar que as passagens R_0 não estão mais expandidas, foram utilizadas nesse segundo estágio as passagens originais. A saída do monoT5 produzida é formada pelos top k_1 candidatos (conjunto R_1).

Assim como o monoT5 pode ser custoso quando temos muitos textos candidatos, o terceiro estágio (H_2), chamado de duoT5, também sofre com esse problema. Sendo assim, o monoT5 reduz ainda mais a quantidade de candidatos que são fornecidos ao duoT5, aqui chamado de reranqueador pairwise. Nessa abordagem pairwise, o reranqueador considera um par de candidatos de uma query e estima a probabilidade que um texto i seja mais relevante que um texto j para ela. Isso é feito com todos os candidatos de uma query resultantes do estágio anterior, dessa forma, o resultado obtido é: a combinação par a par entre todos os candidatos R_1 de uma query, ou seja, $(k_1 * k_1 - 1)$ comparações pairwise. Com isso, cada candidato de uma query vai ter $k_1 - 1$ scores $s_{i,j}$ e é necessário se utilizar de uma função de agregação para combiná-los em um único score para que então os top k_2 candidatos possam ser retornados como os resultados R_2 da arquitetura de busca.

Normalmente, do primeiro H_0 para o segundo H_1 estágio são fornecidos $k_0 = 1000$ candidatos por query. Com isso, o monoT5 reranquea esses 1000 candidatos por query e fornece os top k_1 candidatos para que o duoT5 também possa realizar seu reranqueamento, porém, foi observado empiricamente no dataset TREC Deep Learning 2020 Track que ao aumentar o número k_1 de candidatos a serem fornecidos para o duoT5 ocorre uma degradação nas métricas de qualidade, nDCG@10 e nDCG@20, para esse dataset. Essa degradação pode ser vista na Tabela 1 a seguir, onde temos a quantidade de candidatos aumentando e as métricas caindo quando utilizamos uma função de agregação sym-sum [7]. Uma suspeita é que a função de agregação utilizada seja o problema.

Tabela 1: Resultados usando duo-base reranking n docs from monoT5-base+d2q+rm3 e função de agregação Sym Sum

Candidatos	NDCG@10	NDCG@20	J@10	J@20
30	0.7308	0.7028	0.9852	0.9130
50	0.7306	0.7024	0.9759	0.9157
100	0.7298	0.6985	0.9778	0.9139
300	0.7293	0.6996	0.9796	0.9130

Dito isso, esse trabalho tem como objetivo replicar alguns resultados obtidos por [7] com diferentes funções de agregação, analisar os resultados e entender se conseguimos, empiricamente, encontrar novas funções de agregação que melhorem os resultados alcançados quando aumentamos a quantidade de candidatos fornecidos ao duoT5.

2 Dataset

A arquitetura apresentada foi avaliada na tarefa de ranqueamento de passagens do TREC 2020 Deep Learning Task [2]. Dentro da área de information retrieval, esse dataset vêm sendo bastante utilizado nos estudos de modelos de ranqueamento ad-hoc quando necessitamos de uma grande quantidade de dados.

O TREC-20 DL Track (passage retrieval) conta com 200 queries, mas apenas as 54 delas que possuem passagens julgadas foram utilizadas no projeto. Essas 54 queries possuem uma média de $\sim$ 210 julgamentos por query, variando de 152 a 368 passagens julgadas por query.

As notas dos julgamentos variam em uma escala de quatro pontos, de 0 a 3 [2]:

- Nota 3 (Perfeitamente relevante): a passagem é dedicada à query e contém a resposta exata.
- Nota 2 (Altamente relevante): a passagem contém algumas informações para responder a query, mas ainda assim a resposta não está totalmente clara.
- Nota 1 (Relacionada): a passagem parecer ser relacionada à query, mas não a responde.
- Nota 0 (Irrelevante): a passagem não tem nada a ver com a query.

Para métricas que binarizam a escala de julgamentos, os julgamentos 3 e 2 são normalmente mapeados para relevante enquanto 0 e 1 são mapeados para irrelevante.

Nos experimentos descritos a seguir, foram utilizados três arquivos:

- collection.tsv: arquivo contendo os ids e os textos das 8841823 passagens do corpus.
- topics.dl20.txt: arquivo contendo os ids e os textos das 200 queries do corpus.
- qrels.dl20-passage.txt: arquivo contendo, para 54 queries, o mapeamento das passagens que foram julgadas e as respectivas notas.

3 Metodologia

Como visto na Figura 1, temos duas etapas de reranqueamento, uma utilizando um reranqueador pointwise e outra um pairwise. A diferença entre os dois se dá, principalmente, em suas saídas. Denominamos **pairwise scores** a representação onde, dada uma query e seus textos candidatos, para cada texto temos a probabilidade dele ser mais relevante para a query do que todos os outros e essa comparação é realizada par a par:

$$P(d_i > d_j | d_i, d_j, q) \tag{1}$$

Para realizar a tarefa de reranqueamento é necessário que possamos ordenar nossos textos de acordo com um determinado score. Logo, precisamos ter para a query um único score associado por texto candidato. Por esse motivo, devemos aplicar uma **função de agregação** nos pairwise scores para que eles sejam

agregados de forma a termos um único score por texto por query. Para esse tipo de score denominamos **pointwise score**. Abaixo, podemos ver um exemplo do que seria um pointwise score:

$$P(Relevância = 1|d_i, q) \tag{2}$$

Para avaliar e experimentar diferentes funções de agregação do DuoT5 é preciso, primeiro, possuir os pairwise scores gerados como saída do modelo. Para obtenção desses resultados, como vimos na seção acima, é necessário que o DuoT5 receba como entrada pares de textos reranqueados pelo MonoT5 e a query na qual será avaliada os textos. Sendo assim, podemos dividir a metodologia em três etapas:

1. Obter os resultados do MonoT5 para cada query;
2. Obter os resultados do DuoT5 para cada query e par de textos possível;
3. Avaliar diferentes funções para agregar os resultados.

Reutilizamos os resultados do MonoT5[1] que já haviam sido previamente calculados pelo Rodrigo Nogueira.

Com esses resultados em mãos, precisávamos gerar a saída do DuoT5[2] nessa segunda etapa. Contudo, a biblioteca utilizada[3] não trazia como retorno os pairwise scores, apenas os pointwise scores, que já haviam sido processados por uma função de agregação. Assim, tivemos que realizar algumas alterações no código fonte para que fosse possível receber como retorno os pairwise scores.

Após as modificações, fizemos uso do Google Colab com máquinas com GPUs P100, para realizar a inferência dos pairwise scores. Essa processo foi realizado para 30, 50, e 100 candidatos, o desafio maior foi executar o modelo para 300 candidatos. Primeiramente, tentamos processar todas as queries e textos de uma única vez, processo esse que levaria cerca de 16 horas. Contudo, pouco depois de ter processado metade das queries, o notebook desconectava e, com sua reinicialização, perdíamos o que já havia sido processado. Dessa forma, foi necessário separar as queries em dois notebooks diferentes e rodá-los de forma separada. Com isso, após cerca de 8 horas, todos os textos para todas as queries já haviam sido processados e, dessa maneira, salvamos os resultados obtidos.

Com os resultados, passamos a avaliar uma série de funções de agregação e também analisamos mais a fundo os pairwise score obtidos. Todos os códigos e bases utilizadas estão disponibilizadas em nosso repositório do GitHub[4].

[1] `https://huggingface.co/castorini/monot5-base-msmarco`

[2] `https://huggingface.co/castorini/duot5-base-msmarco`

[3] `Pygaggle:https://github.com/castorini/pygaggle/blob/master/pygaggle/rerank/transformer.py`

[4] `https://github.com/leobavila/ia376e_projeto_final`

4 Experimentos

4.1 Exploração dos pairwise scores

Dado que o DuoT5 estima a probabilidade que um texto i seja mais relevante que um texto j para uma determinada query, partimos da premissa que, ao estimar a probabilidade do texto j ser mais relevante que o texto i para a mesma query, encontraríamos algo próximo à probabilidade complementar, ou seja, esperamos que haja coerência no resultado do modelo em relação a qual texto é mais relevante, independente se estamos comparando d_i com d_j ou d_j com di.

$$\text{Se, } P(d_i > d_j | d_i, d_j, q) = p$$

$$\text{Esperamos que, } P(d_j > d_i | d_i, d_j, q) \approx 1 - p$$

A partir dessa hipótese, iniciamos uma exploração dos pairwise scores obtidos com objetivo de verificar se ela realmente ocorria. Para o caso onde essa hipótese não se concretizou denominamos de ***flip***, ou seja, em um primeiro momento o modelo indica que o texto i é mais relevante que o texto j, mas, ao inverter a posição dos textos na predição, o modelo indica que o texto j é mais relevante que o texto i. Para exemplificar, iremos apresentar um caso onde ocorreu um *flip*:

Para a $query\_id = 768208$, que apresenta a seguinte busca: "what is mamey"e para os textos $doc\_id_i = 1008083$ e $doc\_id_j = 112760$ temos que os resultados do DuoT5 são:

$$P(doc\_id_i > doc\_id_j | doc\_id_i, doc\_id_j, query\_id) = 1.68e^{-6} \tag{3}$$

$$1 - P(doc\_id_j > doc\_id_i | doc\_id_j, doc\_id_i, query\_id) = 0.999 \tag{4}$$

Assim, podemos notar que na Equação 3, onde estimamos a relevância do texto i em relação ao texto j, temos um score bastante baixo, indicando pouca relevância. Entretanto, quando estimamos a probabilidade complementar do texto j ser mais relevante que o texto i obtemos um score alto, indicando que o texto i é mais relevante que o texto j.

Realizamos a contagem de *flips* ocorridos em cada texto em relação aos demais por query. Com isso, calculamos a taxa média de *flips* por query, sendo representada pela equação 5:

$$\overline{x}_{query} = \frac{1}{n-1} * \sum_{1}^{n} \frac{\#flips}{n}, \tag{5}$$

onde n é o número de textos.

Na Figura 2, podemos observar que, ao aumentar a quantidade de textos a serem reranqueados, nossa distribuição de *flips* se altera bastante, chegando à queries com uma taxa de *flips* de quase 80% para 300 textos. Tal evidência corrobora com a degradação dos resultados ao aumentar a quantidade de textos

a serem reranqueados e pode ser um possível fator de ruído ao aplicar a função de agregação.

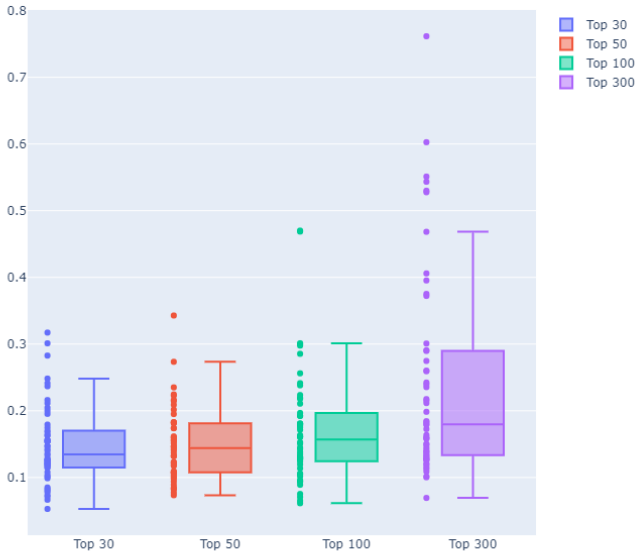

Figura 2: Distribuição das taxas médias de flip por query para diferente quantidades de textos a serem reranqueados.

4.2 Novas funções de agregação e resultados

Com essa nova informação a respeito dos *flips* que ocorrem nos pairwise scores, entendemos que, para amenizar o efeito nas métricas decorrente desse ruído, devemos formular funções de agregação no sentido de penalizar os casos onde ocorrem os *flips* ou grandes diferenças entre a probalidade de um texto i ser mais relevante que um texto j e o complemento da probabilidade do texto j ser mais relevante que o texto i.

4.2.1 Sym-Sum-Log

A função Sym-Sum-Log é chamada assim pois realiza a soma simétrica dos logs, ou seja, soma o log da probabilidade do texto i ser mais relevante em relação aos outros textos para determinada query, bem como a probabilidade complementar dos outros textos serem mais relevantes que o texto i. Essa função foi apresentada em [7], mas não foi testada dessa maneira.

$$s_i = \sum_{j \in J_i} (\log p_{i,j} + \log (1 - p_{j,i})), \tag{6}$$

Onde J_i representa o conjunto de textos a comparar o texto i, excluindo ele mesmo.

Na Tabela 2 podemos ver os resultados obtidos usando tal função. Ao comparar com os resultados obtidos na Tabela 1 verificamos que obtivemos um aumento em relação ao NDCG@10 e um crescimento da métrica à medida que se aumentou o número de textos candidatos. Entretanto, tal crescimento não se manteve em relação ao NDCG@20.

Tabela 2: Resultados da função de Agregação Sym-Sum-Log

Candidatos	NDCG@10	NDCG@20	J@10	J@20
30	0.7309	0.7015	0.9852	0.9130
50	0.7324	0.6983	0.9796	0.9120
100	0.7337	0.6992	0.9796	0.9130
300	0.7338	0.6996	0.9796	0.9139

4.2.2 Loop-Truncation-Aggregation

A medida que aumentamos a quantidade de textos candidatos a serem reranqueados, mais textos não relevantes à query estarão presentes. A ideia por trás desse algoritmo de agregação é ir diminuindo, aos poucos, a quantidade de textos a serem considerados na função de agregação. No Algoritmo 1 podemos visualizar o seu funcionamento para 300 candidatos.

Algorithm 1 Algoritmo Loop-Truncation-Aggregation

```
pairwise_scores ← DuoT5(query, list_of_candidates)
pointwise_scores ← aggregate_function(pairwise_scores)
for cut in [200, 100, 50]:
    pairwise_scores ← truncate_n(pairwise_scores, pointwise_scores, n=cut)
    pointwise_scores ← aggregate_function(pairwise_scores)
```

Onde *truncate_n* é a função que seleciona, com base no resultado agregado (pointwise), os n textos a serem mantidos no pairwise para a próxima iteração, e a função *aggregate_function* pode ser qualquer função que transforme os pairwise scores em pointwise scores. Em nosso experimento usamos a função Sym-Sum-Log como função de agregação.

Tabela 3: Resultados do Algoritmo Loop-Truncation-Aggregation para 300 candidatos

Candidatos	NDCG@10	NDCG@20	J@10	J@20
300	0.7295	0.7002	0.97963	0.914815

Como vemos na Tabela 3, o resultado para 300 candidatos foi ligeiramente superior ao obtido na Tabela 1. Além disso, esse algoritmo é parametrizado, o que limita sua aplicação para diferentes quantidades de textos.

4.2.3 Out-of-flip

Baseado no conceito de *flip* e na ideia de que uma passagem que apresenta *flip* com a passagem de última posição retornada pelo MonoT5 não deveria ser levada em consideração na função de agregação, desenvolvemos a "função"Out-of-Flip. Nela, primeiro geramos o conjunto de textos que não tiveram *flip* com o pior texto retornado pelo MonoT5 (Equação 7). Usamos então esses textos para gerar o pointwise score para um texto i (Equação 8).

$$D_{not\_flip} = d_i \ni F_{mono\_worst} \tag{7}$$

$$s_i = \sum_{j \in D_{not\_flip}} (\log p_{i,j} + \log (1 - p_{j,i})) \tag{8}$$

Onde F_{mono_worst} é o conjunto de textos nos quais ocorreram *flip* com o texto de última posição retornado pelo MonoT5.

Tabela 4: Resultados da função de Agregação Out-of-flip

Candidatos	NDCG@10	NDCG@20	J@10	J@20
30	0.7298	0.6936	0.9833	0.8915
50	0.7310	0.6971	0.9796	0.9102
100	0.7331	0.6990	0.9796	0.9120
300	0.7332	0.6995	0.9796	0.9139

Na Tabela 4 podemos verificar que obtivemos resultados melhores para maiores quantidades de candidatos do que aqueles obtidos na Tabela 1. Além disso, um importante resultado obtido foi que ambas as métricas (NDCG@10 e NDCG@20) cresceram a medida que se aumentou a quantidade de textos candidatos.

4.2.4 Proportional-To-Score-Distance

Seguindo a ideia de que os scores devem se manter coerentes ao alterar a ordem de qual texto queremos estimar a relevância, desenvolvemos a função Proportional-to-Score-Distance (Equação 9). Nessa função, primeiro calculamos a distância absoluta entre o score gerado para o texto i ser mais relevante que o j e o gerado pelo complemento do score de o texto j ser mais relevante que o i. Após isso, subtraímos 1, pois quanto menor a distância, mais peso queremos dar para essa comparação e quanto maior, mais queremos penalizar, multiplicando por um fator menor. Por fim, multiplicamos o resultado do passo anterior pelo log, realizando esses passos para todos os textos.

$$s_i = \sum_{j \in J_i} (1 - |p_{i,j} - (1 - p_{j,i})|) * \log p_{i,j} \tag{9}$$

Tabela 5: Resultados da função de Agregação Proportional-To-Score-Distance

Candidatos	NDCG@10	NDCG@20	J@10	J@20
30	0.7332	0.7042	0.9833	0.9130
50	0.7322	0.7003	0.9815	0.9148
100	0.7352	0.7001	0.9815	0.9120
300	0.7334	0.6999	0.9815	0.9130

Na Tabela 5 podemos observar que obtivemos resultados melhores do que os obtidos na Tabela 1 para quase todas as quantidades de candidatos, incluindo a melhor métrica de nDCG@10 (candidatos = 100) e nDCG@20 (candidatos = 30). Apesar disso, o crescimento constante das métricas não ocorreu.

5 Conclusão

Nesse projeto, nós testamos uma série de diferentes funções de agregação para consolidar os pairwise scores gerado pelo DuoT5 em pointwise scores. Além disso, nos aprofundamos em entender o problema de flips e em como remover essa causa de ruído dos resultados finais. Podemos resumir nossas conclusões em:

- Foram encontradas funções que conseguiram melhorar os resultados obtidos até então, como Sym-Sum-Log e Proportional-To-Score-Distance;
- Além disso, obtivemos com a função Out-of-Flip um resultado monotônico para o NDCG@10 e NDCG@20, nos quais o crescimento das métricas a medida que cresce a quantidade de candidatos é coerente com o que era esperado de desempenho pelo DuoT5;
- Após análises dos pairwise scores, encontramos que em alguns casos o modelo se contradiz em relação a qual texto é mais relevante que o outro. Denominamos esse comportamente de "flip"e verificamos que o mesmo aumenta a medida que aumentamos o número de candidatos, sendo, portanto um fator de ruído que pode estar prejudicando os resultados do modelo;

6 Trabalhos Futuros

Como próximos passos podemos citar a abertura de uma solicitação de mudança (*Pull Request*) para o repositório do PyGaggle no GitHub para que a classe do DuoT5 passe a retornar os pairwise scores, de maneira opcional. Realizar novos

experimentos utilizando as funções de agregação que apresentaram melhores resultados, mas utilizando o DuoT5 3B. Por fim, realizar o treinamento do modelo DuoT5 adicionando uma nova classe para quando não há uma relação de relevância entre os textos, a fim de prevenir o problema de "flip"encontrado.

Referências

[1] Nima Asadi and Jimmy Lin. Effectiveness/efficiency tradeoffs for candidate generation in multi-stage retrieval architectures. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, page 997–1000, New York, NY, USA, 2013. Association for Computing Machinery.

[2] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track. *CoRR*, abs/2102.07662, 2021.

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[4] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. Pretrained transformers for text ranking: BERT and beyond. *CoRR*, abs/2010.06467, 2020.

[5] Rodrigo Nogueira. From doc2query to doctttttquery. 2019.

[6] Rodrigo Nogueira and Kyunghyun Cho. Passage re-ranking with BERT. *CoRR*, abs/1901.04085, 2019.

[7] Ronak Pradeep, Rodrigo Nogueira, and Jimmy Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *CoRR*, abs/2101.05667, 2021.

[8] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683, 2019.

[9] Stephen Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, pages 109–126. Gaithersburg, MD: NIST, January 1995.