ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT

论文链接：https://arxiv.org/pdf/2004.12832.pdf

1. 背景

这篇文章是针对大规模的检索召回而言的。对于精排阶段，自然可以把query-document pair输入进一个复杂的网络，计算出一个relevance score出来，但是召回阶段需要的是快，所以不能这样做。

在神经信息检索中，有如下几种模式：

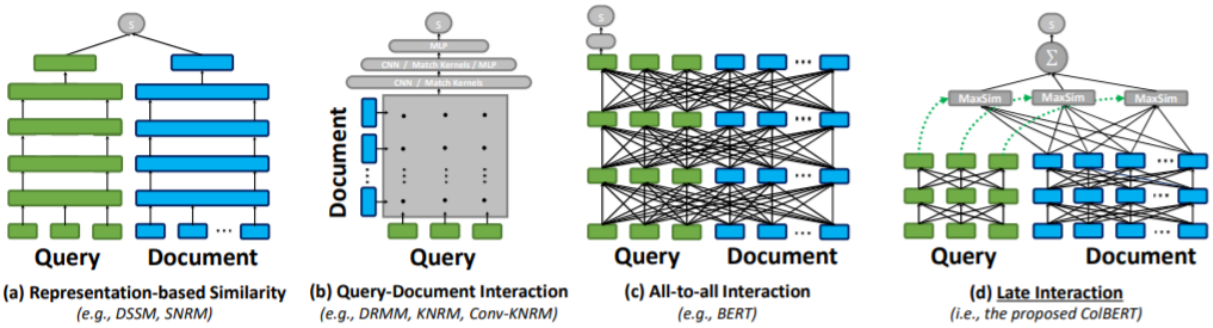

Figure 2: Schematic diagrams illustrating query–document matching paradigms in neural IR. The figure contrasts existing approaches (sub-figures (a), (b), and (c)) with the proposed late interaction paradigm (sub-figure (d)).

(a). 双塔分离，不做交互

(b). 在底部就对每个token做交互

(c). 拼接query、document，输入BERT进行self-attention操作

(d). ColBERT虽然是对query和document**分开**做的encoding，但是后面又用了一个"cheap yet powerful"的**交互**来把握更精细的相似度。这样，document embedding还是可以**离线**算好（900万个MSMARCO passage可以在三小时内建好索引，存入faiss）。在线上用query embedding进行相似度检索即可；同时，还保留了query和document的交互，可谓是达到了效率和准确率的平衡。所以文章标题叫"**Efficient and Effective**".

2. 模型结构

2.1 Query & Document Encoder

首先，需要对预训练好的BERT进行**微调**。使用正负样本对<q, d+, d->，分别对正负样本都用ColBERT的方法（MaxSim）计算相似度score，然后使用cross entropy loss进行梯度回传 (label: [0])。

对query和document都分别过预训练+微调好的BERT，输出的每个token embedding。这个embedding自然是分别考虑了query和document的上下文的。这就是题目中"**contextualized**"的由来。

具体地，query和document共享BERT参数，只是在句子前面加上[Q]和[D]来标记是query还是document。经过BERT之后再经过一个线性层，把维度降成m维。然后做normalize（为了可以直接计算点积来表示相似度）。

$$E_q := \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[Q]q_0q_1\ldots q_l\#\#\ldots\#\text{”})\ )\ ) \quad (1)$$

$$E_d := \texttt{Filter}(\ \texttt{Normalize}(\ \texttt{CNN}(\ \texttt{BERT}(\text{“}[D]d_0d_1\ldots d_n\text{”})\ )\ )\ ) \quad (2)$$

2.2 迟交互

对于每个query token，都去计算它和document的所有token的相似度最大值，最终的score就是所有token的这个最大值之和：

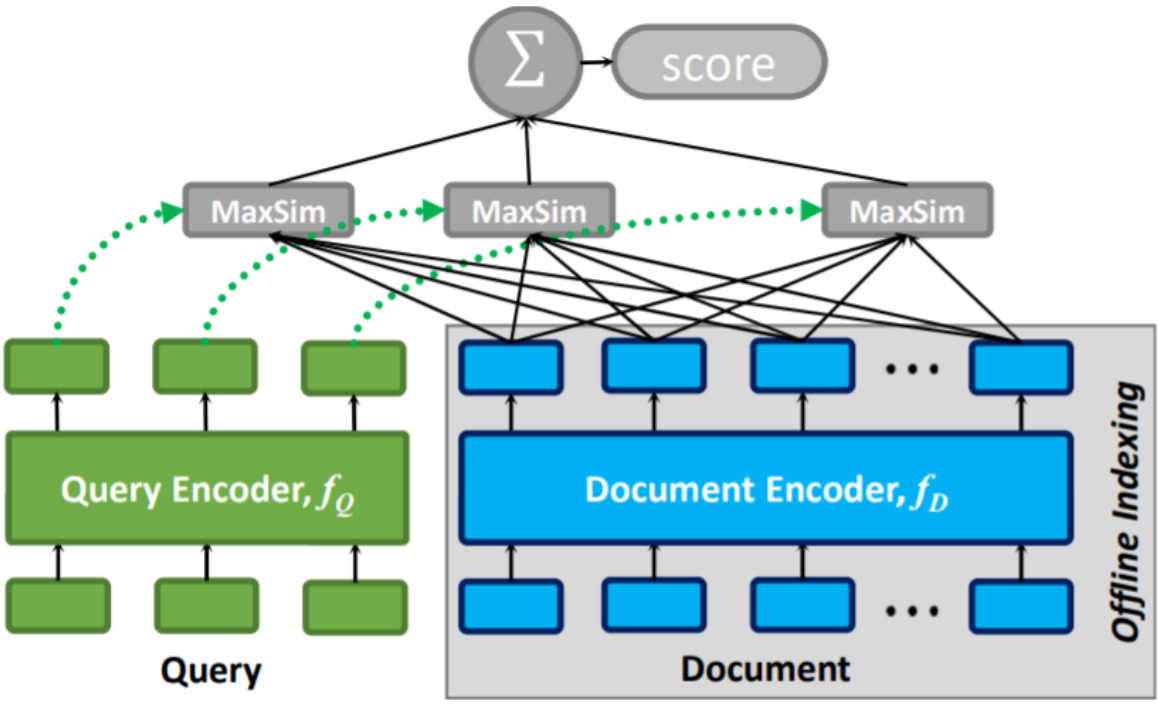

Figure 3: The general architecture of ColBERT given a query q and a document d.

使用MaxSim是因为这样非常"pruning-friendly". 在实际召回中，我们只需要从大规模的语料库(e.g. N = 1000万)中找K个最为相似的(e.g. K = 1000). 那么，自然是不用全部计算N个document的打分。具体的做法是：

- 把query的 N_q 个token embedding全部输入faiss进行查询，每个token embedding查出来 k' = k/2个最相似的document token embedding。共有 $N_q * k'$ 个。
- 找到这$N_q * k'$个document token所对应的document，当然这会有重复，所以需要去重。
- 对去重之后的document可以用精排的方式来确定要返回的top k个。