本文对预训练模型在召回(retrieval), 排序(re-ranking)，以及其他部分的应用做一个总结，参考学长们的综述：https://arxiv.org/pdf/2111.13853.pdf

1. 背景

搜索任务就是给定一个query或者QA中的question，去大规模的文档库中找到相似度较高的文档，并返回一个按相关度**排序**的**ranked** list。由于待训练的模型参数很多（增加model capacity），而专门针对检索任务的有标注数据集较难获取，所以要使用预训练模型。

2. 检索模型的分类

检索的核心，在于计算query和document的**相似度**。依此可以把信息检索模型分为如下三类：

- 基于统计的检索模型。使用exact-match来衡量<query,document>相似度,考虑的因素有query中的词语在document中出现的词频TF、document长度（惩罚长文本，例如一个词在300页的文章中出现过2次远远不如一个词在一小段微博动态里出现过两次）、逆文档词频IDF（惩罚在所有文档中都出现过很多次的词，例如"的"）。代表性的模型是BM25，用来衡量一个term在doc中的重要程度，其公式如下：

$$\sum_{i}^{n} IDF(q_i) \frac{f(q_i, D) * (k1 + 1)}{f(q_i, D) + k1 * (1 - b + b * \frac{fieldLen}{avgFieldLen})}$$

惩罚长文本、对词频做饱和化处理

实际上，BM25是检索模型的强baseline。基于exact-match的检索模型是召回中必不可少的一路。

- Learning-to-Rank模型。这类模型需要手动构造特征，包括query端特征，如query类型、query长度（还可以加入意图slot?); document端特征（document长度，Pagerank值）；query-document匹配特征（BM25值，相似度，编辑距离等）。其实，在现在常用的深度检索模型中也经常增加这种人工构造的特征。根据损失函数又可分为pointwise（简单的分类/回归损失）、Pairwise(triplet hinge loss，cross-entropy loss)、Listwise。
- 深度模型。使用query和document的embedding进行端到端学习。可以分为representation-focused models（用双塔建模query和document，之后计算二者相似度，双塔之间无交互，用于召回）、interaction-focused models（金字塔模型，计算每个query token和每个document token的相似度矩阵，用于精排。精排阶段还可增加更多特征，如多模态特征、用户行为特征、知识图谱等）

3. 预训练模型在倒排索引中的应用

基于倒排索引的召回方法仍是在第一步召回中必不可少的，因为在第一步召回的时候我们面对的是海量的文档库，基于exact-match召回速度很快。但是，其模型capacity不足，所以可以用预训练模型来对其进行模型增强。

3.1 term re-weighting

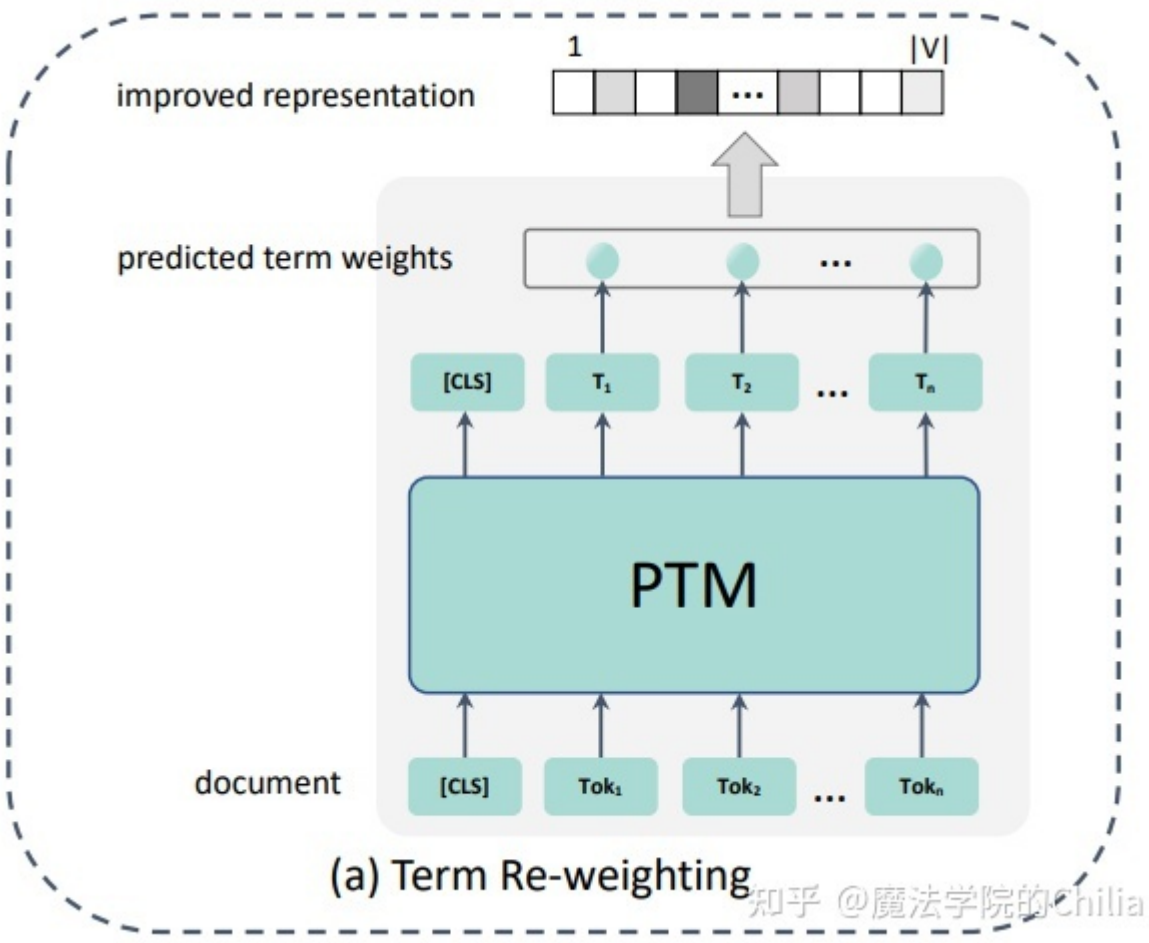
代表论文：**DeepCT** (Deep Contextualized Term Weighting framework: Context-Aware Sentence/Passage Term Importance Estimation For First Stage Retrieval).

普通的exact-match中衡量一个词在query/document中的重要程度就是通过词频(TF)或者TFIDF，或者TFIDF的改进版本--BM25，例如在建立倒排索引的时候，每个term在不同document的重要程度就是用TF来衡量的。但是，一个词在两个document中出现频率相同，就说明这个词在两个document中同样重要吗？其实词的重要程度比词频要复杂的多。所以，可以使用contextualized模型，例如BERT，Elmo等获得每个词的**上下文**表示，然后通过简

单的线性回归模型得到每个词在document中的重要程度。文档真实词语权重的估计如下，这个值作为我们训练的label：

$$QTR_{t,d} = \frac{|Q_{d,t}|}{|Q_d|}$$

其中，Q_d 是与文档 d 相关的查询问题的集合；$Q_{d,t}$ 是包含词语 t 的查询问题集合 Q_d 的子集；$QTR_{t,d}$ 是文档 d 中词语 t 的权重。$QTR_{t,d}$的取值范围为[0，1]，以此为label训练。这样，我们就得到了一个词在document中的重要程度，可以替换原始TF-IDF或BM25的词频。对于query，也可以用同样的方法得到每个词的重要程度，用来替换TFIDF。



3.2 Document expansion

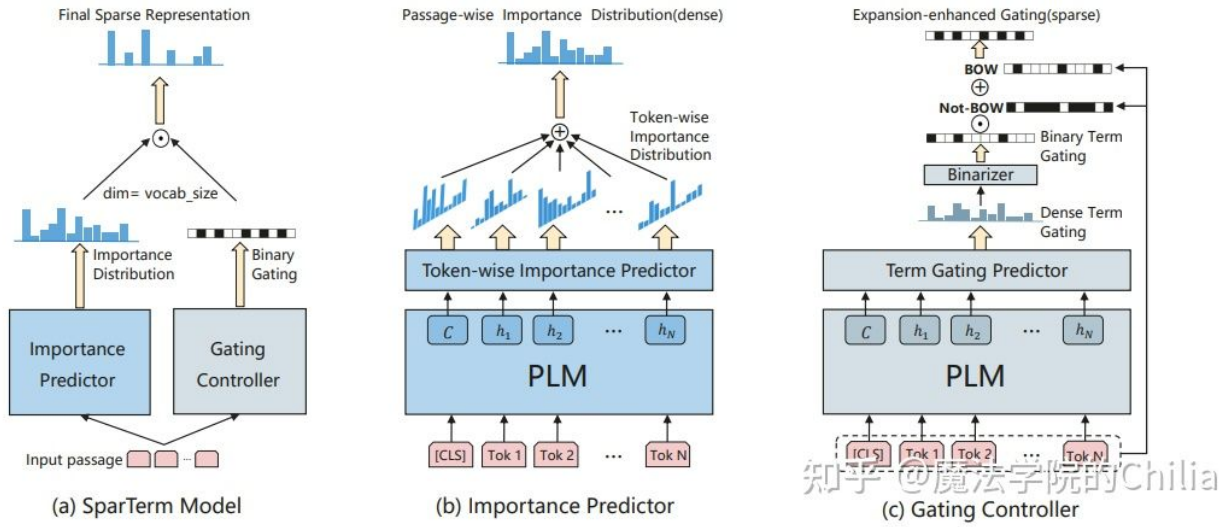
除了去估计不同term在document中的重要程度，还可以直接显式地扩增document，这样一来提升了重要词语的权重，二来也能够召回"词不同意同"的文档（解决lexical-mismatch问题）。例如，可以对T5在query-document对上做微调，然后对每个document做文本生成，来生成对应的query，再加到document中。之后，照常对这个扩增好的document建倒排索引，用BM25做召回。代表工作：docTTTTTquery
https://cs.uwaterloo.ca/~jimmylin/publications/Nogueira_Lin_2019_docTTTTTquery-v2.pdf

同样地，也可以对query进行扩增。例如对于QA中的question，可以把训练目标定为包含答案的句子、或者包含答案的文章title，然后用seq2seq模型训练，再把模型生成的文本加到query后面，形成扩增的query。

3.3 term reweighting + document expansion

那么，我们可不可以同时做term reweighting和document expansion呢？这方面的代表工作是Sparterm: https://arxiv.org/pdf/2010.00768.pdf

此模型分为两部分：重要度预测模块（用来得到**整个vocab上**的重要程度）和门控模块（得到二进制的门控信号，以此来得到最终保留的稀疏token，最终只能保留 λ 个token）。由于重要度是针对整个vocab而言的，所以可以同时实现重要度评估+词语扩增。



重要度预测模块采用了类似MLM的思想，即先用BERT对句子做好contextualized embedding，然后乘上vocab embedding 矩阵

$$E$$

，得到这个词对应的重要度分布：

$$I_i = BERT(h_i)E^T + b$$

这句话整体的重要度分布就是所有词对应的重要度分布取relu（重要度不能是负数），然后加起来的和：

$$I = \sum_{i=0}^{L} Relu(I_i)$$

门控模块和重要度评估模块的计算方法类似，只是参数不再是 E，而是另外的变换矩阵。得到gating distribution G 之后，先将其0/1化为 G' （如果G中元素>threshold则取1，否则取0）；然后得到我们需要保留的词语（exact-match必须保留，还增加一些扩增的token）。

通过端到端的方式训练，训练的损失函数有两个，其中一个就是我们常见的ranking loss，即取<query,正doc,负doc>三元组，然后求对比cross-entropy loss。这里的q', p'都是经过重要度评估模块+门控模块之后得到的句子表征,因此loss反传可以对重要度评估模块和门控模块进行更新。

$$L_{rank}(q_i, p_{i,+}, p_{i,-}) = -\log \frac{e^{sim(q_i', p_{i,+}')}}{e^{sim(q_i', p_{i,+}')} + e^{sim(q_i', p_{i,-}')}}$$

另一个loss专门对门控模块做更新，训练数据是<query, doc>对，对于一个document，先得到其门控向量G, 然后去和实际的query进行对比：

$$L_{exp} = -\lambda_1 \sum_{j \in \{m|T_m=0\}} log(1 - G_j) - \lambda_2 \sum_{k \in \{m|T_m=1\}} logG_k \quad (8)$$

query中没有改词. query中有改词

T为真实query的bag of words

预训练模型在深度召回和精排中的应用

4. 预训练模型在深度召回中的应用

在深度召回中，我们使用Siamese网络生成query/doc的embedding，然后用ANN(approximate nearest neighbor)进行召回。

4.1 基本模型结构

对于query/doc，可以用一个、或者多个embedding表示。

1）query和doc均用一个embedding表示的代表模型:

DPR (Deep Passage Retrieval For Open Domain Question Answering，https://arxiv.org/pdf/2004.04906.pdf) 。DPR使用两个**不同的** BERT-base encoder来对question和海量的document进行表征。在训练时使用正doc和n个负doc，其中n个负doc的采样方法可以是：

- random：从doc库中随机采样
- BM25: 取BM25最高，但不是正样本的doc。即那些虽然有exact-match，但并不是answer的document，作为难负例。
- in-batch负采样：取batch中其他B-1个doc作为负样本。

之后计算交叉熵损失：

n个负样本

$$L(q_i, p_i^+, \underline{p_{i,1}^-, \cdots, p_{i,n}^-}) \quad (2)$$

$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^{n} e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

知乎 @魔法学院的Chilia

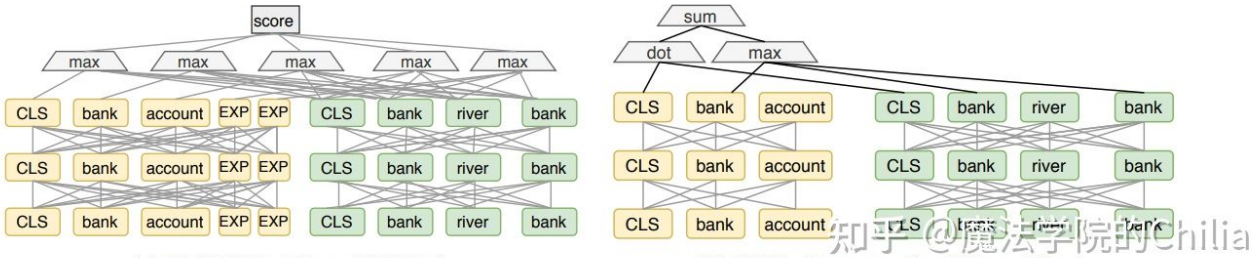
1）query和document均使用多个embedding的代表模型：

1.1) ColBERT:

[魔法学院的Chilia：搜索] SIGIR'20: 基于迟交互的向量检索召回模型ColBERT13 赞同 · 0 评论文章

1.2) COIL (Contextualized Inverted List):

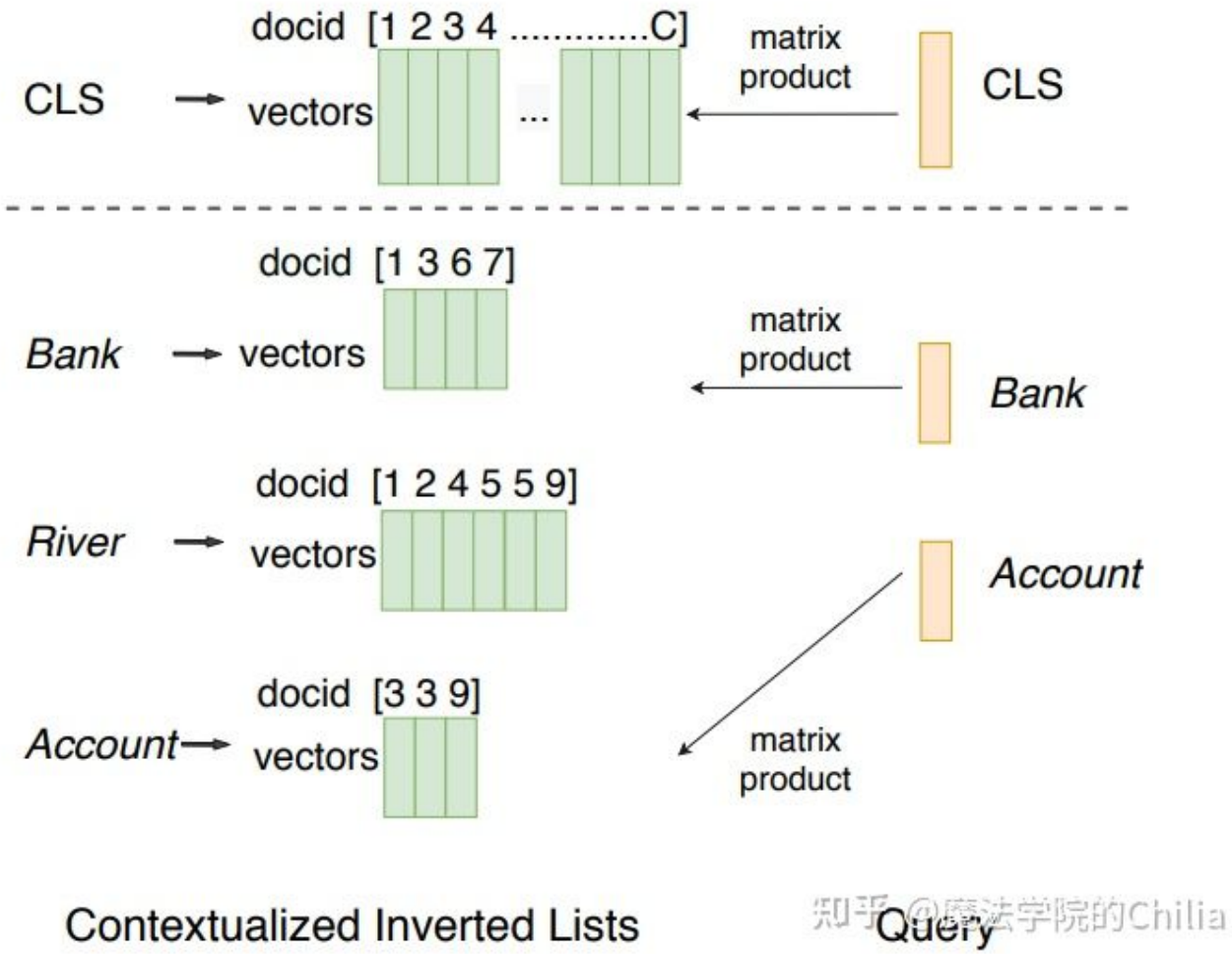
和ColBERT类似，只不过只对那些**精确匹配**的词做交互，而不用对所有的token都做交互，这样进一步提升了效率，且效果和ColBERT几乎等同。同时，为了能够进行semantic匹配，还对query和document的**[CLS]**计算相似度，最终的相似度即为[CLS]相似度和token匹配的maxsim之和。



(c) ColBERT: All-to-All Match　　(d) COIL: Contextualized Exact Match

只对精确匹配的部分计算maxsim。这是因为大部分精确匹配的相似度都要>非精确匹配的相似度，就不必求一遍所有的相似度了。

此方法可以自然的融入倒排索引架构，只需要对每个document都存储一个[CLS] embedding；对所有的token都存储下来包含此token的document id，以及此token在该doc中的上下文表示。



此外一个常见的做法是，用一个embedding去表示query（因为query通常较短、意思集中），用多个embedding去捕捉document的不同子空间的信息。<q,d>相似度即是query和document的每个embedding的点积最大值。

4.2 预训练任务

我们知道，预训练任务和下游任务越相似，模型在下游任务上的表现就越好。所以，应该设计专门针对检索任务的预训练任务。文章Pre-training Tasks for Embedding-based Large-scale Retrieval提出了三个针对检索设计的预训练任务，都是使用Wikipedia的大量无标注数据进行自监督训练。训练数据是都是<query,doc+>样本对，这些样本对的获取方式有三个：

- Inverse Cloze Task(ICT): 从一段话中随机取一句话作为query，其余句子作为document

- Body First Selection(BFS): 从一个wiki page中的第一段随机取一句话作为query，再随机取该页中的任意一段作为document
- Wiki Link Prediction (WLP): 从一个wiki page中的第一段随机取一句话作为query，再随机取另一个**链接**到该页的页中任意一段作为document.

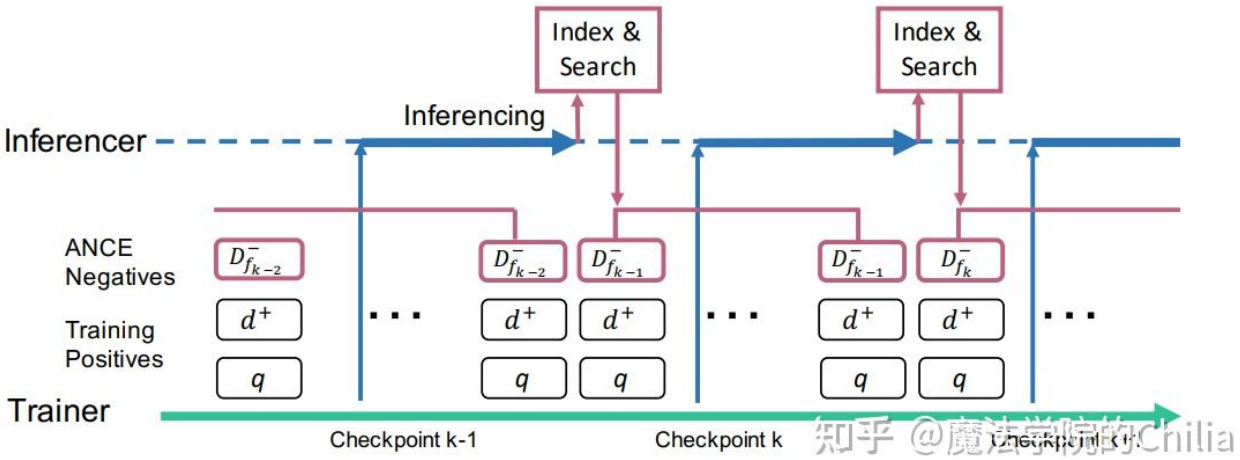
4.3 负采样技术

和精排阶段不同，召回阶段需要人工来做负采样。精排阶段我们只需要以召回但不相关（曝光未点击）作为负例，而召回阶段怎么构造负样本则直接影响着模型性能。

代表论文：ANCE （Approximate Nearest Neighbor Negative Contrastive Learning for dense text retrieval, https://arxiv.org/pdf/2007.00808.pdf)

文中指出，端到端训练的深度检索模型有时效果甚至不如基于exact-match的稀疏检索模型，这是因为过多的使用简单负例（random或者in-batch负采样）没有提供很多信息量，其**梯度范数较小、收敛速度慢**。因此，需要从**全局**找到难负例。

同时，ANN查找的索引是**异步更新**的(asynchronously updated)。如果我们不用难负例采样，那么只需要在训练完成之后建立一次索引，来存储每个document的表征。但是，由于我们要进行难负例采样，所以每次训练的时候我们都需要找到根据当前模型的表征结果去找到最接近的负例。所以，每次在试图找难负例时，都使用最近的一个模型checkpoint来建立索引，然后找到和query最相近的负例作为难负例。这样找到的难负例更加informative，loss更大、梯度也更大，因此可以加速收敛。



思考：为什么要异步更新呢？更新索引需要模型的inference+存储所有document索引，虽然存索引相对效率高一些，但是inference需要在整个document库中全部经过模型的forward pass，这样的计算成本很高。训每个batch都更新一遍索引是不可接受的。所以，只需要隔一段时间用最近的checkpoint更新一下索引即可。（当然了，一种更简单的做法是用另一个训好的模型来选择难负例，但是由于这另一个模型毕竟和我们要训练的模型不同，所以不免要牺牲一些准确率。）

具体的，是用Roberta-base预训练模型来初始化双塔模型，然后先是用BM25做warm-up(用BM25做难负例采样)，之后再通过异步方法更新索引，用正在训练的模型的checkpoint进行难负例采样。

5. 预训练模型在Re-ranking(精排)中的应用

精排阶段可以是多个cascading模型级联构成，数据量越来越少、模型越来越复杂。

5.1 判别模型 (discriminative ranking models)

召回阶段使用的是基于表示的双塔模型，而在精排阶段我们可以使用更为复杂的基于交互的金字塔模型，来让query和document进行交互。代表模型有：

1）MonoBERT (https://arxiv.org/pdf/1910.14424.pdf)

将query和document拼在一起，然后把[CLS]的embedding经过MLP得到relevance score。使用point-wise损失（交叉熵损失），即为简单的分类问题。

2）DuoBERT (https://arxiv.org/pdf/1910.14424.pdf)

把query和正负document同时输入模型（[CLS]+Q+[SEP]+ D_+ +[SEP]+ D_- +[SEP]），然后和MonoBERT一样用[CLS]对应的embedding进行预测，预测的label是0，意为 D_+ 比 D_- 更为相关。此为pair-wise损失。

3) CEDR （Contextualized Embeddings for Document Ranking, CEDR: Contextualized Embeddings for Document Ranking ）

先使用BERT获得query和document中每个词的上下文表征，为了把握不同层的信息，对**L层**都计算相似度矩阵：

$$S_{Q,D} \in \mathbb{R}^{L \times |Q| \times |D|}$$

然后对相似度矩阵采用DRMM或者KNRM的方法来提取交互信息。

长文本处理方法：

由于BERT可接受的最长token长度为512，那么对于特别长的句子该如何解决呢？有两种简单的解决方法：段落分数聚合和段落表示聚合。

1）段落分数聚合

使用一个滑窗来得到不同段的句子，然后对每个段都计算匹配得分。最后的得分聚合可以是first/max/avg. 在训练时也使用一篇文章的不同段落进行训练，在标注label的时候，如果这篇文章为相关，那么其所有段落都标记为相关。当然这样会引入噪声，因为一篇文章虽然相关，但未必其每一段都是相关的。

2）段落表示聚合

将一个长文本拆分成若干<512token的段落之后，对每一段都求其[CLS]表征。那么，长文本的整体表征就是每一段[CLS]表征的聚合。这个聚合方法可以是mean/max/sum等数学方法求得的，也可以是再经过深度模型（FFN/CNN/Transformer）得到的。

注意，在召回阶段对于长文本，一般只能使用max方式进行聚合，因为ANN索引查找（如Faiss）是天然支持max-pooling的。其他聚合方法不适合ANN查找，不能提高召回的效率。在精排阶段，无论使用多么复杂的聚合方法都是可以的。

注：把长文本拆成若干段，就都失去了长距离的依赖。可以使用TransformerXL的思想，缓存上一个segment的隐藏层表示，然后当前segment可以通过自注意力关注到上一个segment的隐藏层。

5.2 生成模型

生成模型假定query和document之间有着随机生成过程（stochastic generative process），例如query-likelihood model就是评估的query有多大可能被document生成出来。

参考资料： COIL：结合稠密检索和词汇匹配的更高效检索模型