中国科学院大学

University of Chinese Academy of Sciences

现 代 信 息 检 索 大 作 业

成员 1：张　兆　202228013229029

成员 2：陈国鑫　2022E8013282125

成员 3：顾颂恩　202228015029012

成员 4：周姿能　202228013229058

成员 5：程　爽　202228013229057

解决方案及代码开源地址：https://github.com/zhangzhao219/UCAS-IR-Project

2022 年 12 月

目录

1 引言

1.1 任务描述

本次大作业要求使用采样后的 TREC 2019 训练数据，在 TREC 2020 Passage Ranking 的子赛道 Passage Re-ranking 进行检索竞赛。在 Passage Re-ranking 子任务中，每个查询提供 1000 篇段落进行重新排名。这 1000 个段落是基于 BM25 检索生成的，没有应用于整个集合的词干。

标准答案集合的评分分为四个等级:（1）完全相关：该段落专用于查询，并包含确切的答案；（2）高度相关：段落提供大量的信息与查询有关，但答案可能有点不清楚，或隐藏在无关的信息中；（3）相关：段落看起来与查询相关，但没有回答查询（提供一些信息与查询有关）；（4）不相关：段落与查询无关。其中根据官网的说明，如果对于二值分类来说，（1）和（2）被认为是相关的，（3）和（4）被认为是不相关的。

1.2 赛题分析

根据我们前期调研，发现应用于 TREC 检索任务的主要模型可以分为三类：（1）“trad”：传统信息检索方法如 BM25、RM3；（2）“nn”：使用深度学习的方法或者词向量如 Duet；（3）“nnlm”：使用大规模预训练语言模型如 Bert、XLNet

通过对比上述的三种方案的评价指标，我们发现大规模预训练语言模型“nnlm”效果整体大于其他神经网络模型“nn”和非神经网络模型“trad”。其中语言模型 BERT 在 Passage Ranking 的两个子赛道中效果都最好。因此我们选择 Bert 作为我们这次大作业的 baseline，并根据信息检索课程所学知识对 baseline 进行改进提高算法性能。

1.3 评价指标

此次比赛所使用的评价指标为在验证集上得到的 NDCG@10，具体计算方法

如下式所示：

$$DCG@10 = \sum_{i=1}^{10} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$$IDCG@10 = \sum_{i=1, rel_i \in |REL_i|}^{10} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

$$NDCG@10 = \frac{DCG@10}{IDCG@10}$$

根据 Trec 官方网站的说明，为了避免不同评价实现方式的差异，官网提供了评测脚本 trec_eval-9.0.7.tar.gz，从而保证评价指标的唯一性。该评测脚本是使用 C 语言编写的，可以在 1 秒钟内返回评测结果，安装使用也十分简单。

下方展示本次实验得到的三个结果的评价结果：

```
zhangzhao@ZhangZhao-PC:/mnt/d/Learning/课程资料/现代信息检索/作业/UCAS-IR-Project/file/trec_eval-9.0.7$ ./
trec_eval -m  ndcg_cut  ../data/2020/2020qrels-pass.txt ../result/result_2020qrels_2022_11_20_11_47_09
ndcg_cut_5              all     0.7247
ndcg_cut_10             all     0.7155
ndcg_cut_15             all     0.6938
ndcg_cut_20             all     0.6718
ndcg_cut_30             all     0.6445
ndcg_cut_100            all     0.6144
ndcg_cut_200            all     0.6332
ndcg_cut_500            all     0.6525
ndcg_cut_1000           all     0.6589
zhangzhao@ZhangZhao-PC:/mnt/d/Learning/课程资料/现代信息检索/作业/UCAS-IR-Project/file/trec_eval-9.0.7$ ./
trec_eval -m  ndcg_cut  ../data/2020/2020qrels-pass.txt ../result/result_2020qrels_2022_11_21_05_01_28
ndcg_cut_5              all     0.6809
ndcg_cut_10             all     0.6739
ndcg_cut_15             all     0.6585
ndcg_cut_20             all     0.6407
ndcg_cut_30             all     0.6321
ndcg_cut_100            all     0.6175
ndcg_cut_200            all     0.6399
ndcg_cut_500            all     0.6551
ndcg_cut_1000           all     0.6590
zhangzhao@ZhangZhao-PC:/mnt/d/Learning/课程资料/现代信息检索/作业/UCAS-IR-Project/file/trec_eval-9.0.7$ ./
trec_eval -m  ndcg_cut  ../data/2020/2020qrels-pass.txt ../result/result_2020qrels_2022_11_21_05_05_06
ndcg_cut_5              all     0.7912
ndcg_cut_10             all     0.7615
ndcg_cut_15             all     0.7444
ndcg_cut_20             all     0.7279
ndcg_cut_30             all     0.7070
ndcg_cut_100            all     0.6681
ndcg_cut_200            all     0.6801
ndcg_cut_500            all     0.6924
ndcg_cut_1000           all     0.6947
```

图 1 评价结果

从图 1 可以看出，实验得到的三个结果的 NDCG@10 分别达到了 0.7155，0.6739 和 0.7615，三个结果的含义将在后续进行说明。

2 实现方案

2.1 技术原理

通过广泛调研阅读，我们尝试了多种预训练模型和后续的结构进行微调，并且在学习率参数调节上尝试学习率预热方法。本小节首先介绍我们采用的核心损失函数 Re-ranker，随后介绍我们使用的三种微调结构，包括 ColBERT 和使用 Bert 输出层的两种输出输出分数，最后介绍 warmup 学习率预热方法。

2.1.1 Re-ranker

Re-ranker 是一个交叉编码器，它重新排列上一阶段检索模型的前 k 个结果，以查询 q 和段落 d 的串联作为输入，并输出一个实值分数 $\theta(q,d)$。给定标记的正对 (q^+,d^+) 和检索的 top-k 预测中随机抽取的 $n-1$ 个强负样本段落，采用列表级损失来训练 re-ranker。其损失函数为：

$$-\log \frac{\exp(\theta(q^+,d^+))}{\exp(\theta(q^+,d^+))+\sum_{i=1}^{n-1}\exp(\theta(q^+,d_i^-))}$$

对于我们的检索任务来说，数据集中并没有$n-1$ 个强负样本段落，对于每一个查询来说仅有一个正样本和一个负样本。因此上式也可以看作退化成 Softmax 损失函数。

2.1.2 BERT

自 2018 年谷歌提出 BERT 以来，预训练语言模型在自然语言处理领域取得了很大的成功，在多种 NLP 任务上取得了 SOTA 效果。BERT 本质上是一个基于 Transformer 架构的编码器，其取得成功的关键因素是利用多层 Transoformer 中的自注意力机制（Self-Attention）提取不同层次的语义特征，具有很强的语义表征能力。如图 2 所示，BERT 的训练分为两部分，一部分是基于大规模语料上的预训练（Pre-training），一部分是在特定任务上的微调（Fine-tuning）。

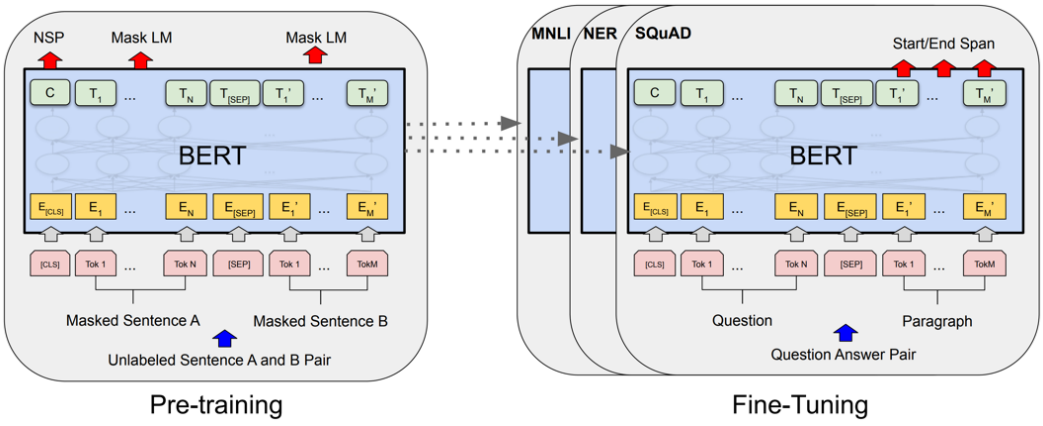

图 2 Bert 的两阶段训练

Bert 相关预训练模型的输出由四部分组成，其中包括模型各层输出的隐藏状态和序列的第一个 token 的最后一层的隐藏状态（pooler output，是由线性层和 Tanh 激活函数进一步处理产生的）。其中模型最后一层输出的隐藏状态（last_hidden_state）通常用于命名实体识别，而 pooler output 的输出通常用于句子分类。因此我们将两种形式的输出作为我们的两种模型微调的结构，后续进行检索实验。

2.1.3 ColBERT

在预训练模型兴起之后，full interaction 的形式变得非常常见，即把 query 和 passage 都输入预训练模型，通过 attention 在整个神经网络的每一层都进行交互计算，实验结果也证明这样的相似度计算方式非常有效。但缺点是计算速度慢，对于低延迟的应用场景不适用。更严重的问题是，passage 无法进行线下预计算和建立索引。因此，这种方案效果虽好，但不能应用在大规模文本检索任务上。原因在于一个 q 需要和大量的 d 进行匹配，计算量和文档个数直接相关，在线上对计算资源依赖大。因此以 ColBERT 等模型提出了所谓的 late interaction。前半程先用各种方法进行预计算编码得到 query 和 passage 的向量表示，而不是交互信息，然后在后半程对编码后的向量做 full interaction，达到性能与速度折衷的目的。Colbert 把表示层尽量离线算出，减少在线实时计算量，只在最后层进行 MaxSim 计算，并且是不需要训练的固定计算，也就是说，这一部分需要的算力资源不多。ColBERT 的主要思想是对 query 与 passage 在 token-level 的编码进行

匹配计算，并通过 MaxSim 运算符取出最大值并求和作为最终的分数。具体公式如下：

$$E_q = Normalize\left(CNN\left(BERT("[Q]q_0q_1 \dots q_l[mask][mask] \dots [mask])\right)\right)$$

$$E_d = Filter\left(Normalize(CNN\left(BERT("[Q]d_0d_1 \dots d_n)\right)\right)$$

其中E_q和E_d不是一个向量，而是对每个 token 编码的向量组，CNN 的含义是 linear 层的降维，Filter 的含义是去掉标点符号的 token 表示。相关性分数的计算：

$$S_{q,d} = \sum_{i \in \|E_q\|} max_{j \in \|E_d\|} E_{q_i} \cdot E_{d_j}^T$$

即对 query 中的每个 token 与 passage 中的所有 token 计算 Sim 值并取出最大值，再将其求和作为最终分数。直观上感觉这种算法与 BM25 等文本匹配算法类似，但计算粒度更为精细。

ColBERT 的网络架构如图 3 所示：

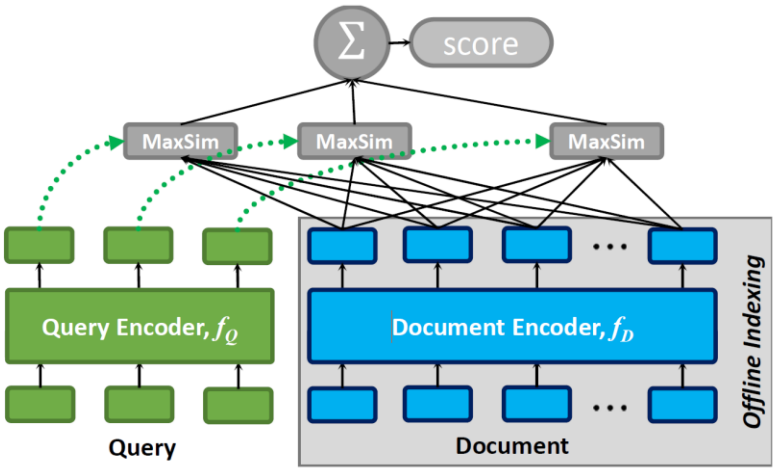

图 3 ColBERT 网络架构图

2.1.4 Warmup

Warmup 是针对学习率优化的一种策略，主要过程是：在预热期间，学习率从 0 线性（也可非线性）增加到优化器中的初始预设值，之后使其学习率从优化器中的初始值线性降低到 0。如图 4 所示：

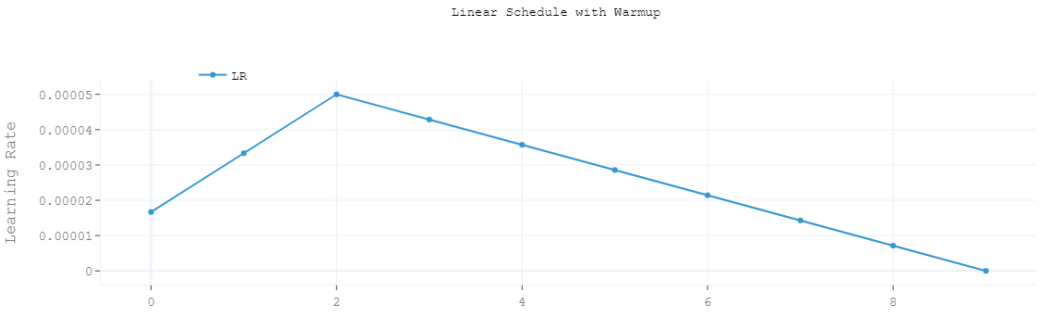

图 4 Warmup 训练技巧示意图

由于刚开始训练时，模型的权重是随机初始化的，此时若选择一个较大的学习率，可能带来模型的不稳定或振荡。选择 Warmup 预热学习率的方式，可以使得开始训练的几个 step 内学习率较小，在预热的小学习率下，模型可以慢慢趋于稳定，等模型相对稳定后再选择预先设置的学习率进行训练,，使得模型收敛速度变得更快，模型效果更佳。同时在模型训练的后期，模型已经基本训练稳定，此时使用高学习率也可能产生震荡的现象，采用更小的学习率进行训练可以使得模型尽可能接近最优点。

2.2 预训练模型

我们在 huggingface 上寻找合适的开源预训练模型作为我们的主干网络，总共找到了三种不同类型的预训练模型共七种，如表 1 所示：

表 1 选取的预训练模型

类型	模型名称	模型网站
C	albert-base-v2	https://huggingface.co/albert-base-v2
	bert-base-uncased	https://huggingface.co/bert-base-uncased
	electra-base-discriminator	https://huggingface.co/google/electra-base-discriminator
M	simlm-base-msmarco	https://huggingface.co/intfloat/simlm-base-msmarco
	cocodr-base-msmarco	https://huggingface.co/OpenMatch/cocodr-base-msmarco
R	cotmae_base_msmarco_reranker	https://huggingface.co/caskcsg/cotmae_base_msmarco_reranker
	simlm-msmarco-reranker	https://huggingface.co/intfloat/simlm-msmarco-reranker

其中，模型类型为 C 指这种预训练模型只在普通的预料上进行预训练过，与 Re-ranking 任务甚至 msmarco 数据集都没有任何联系；模型类型为 M 指该预训练模型在 msmarco 数据集上经过训练，模型类型为 R 指该预训练模型是使用 msmarco 数据集并在 Re-ranking 任务上进行的预训练。

下面介绍三种后续使用的预训练模型。

2.2.1 ELECTRA

ELECTRA 采用 RTD 进行预训练。ELECTRA 的作者分析了 Bert 模型的 MLM 预训练任务的缺点，提出了 Replaced Token Detection 预训练任务，其训练过程类似 GAN，如图 5 所示。ELECTRA 由两个部分组成，第一部分是生成器，生成器将句子中的部分单词进行替换，例如图中将 painting 替换成了 car。第二部分是判别器，判别器用于判断一个句子中每一个单词是否被替换，训练的过程会预测所有的单词，比 BERT 更为高效。

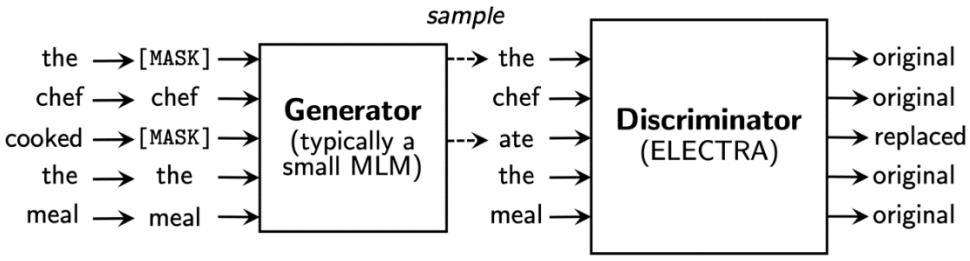

图 5 ELECTRA 结构

2.2.2 SimLM

一、预训练阶段

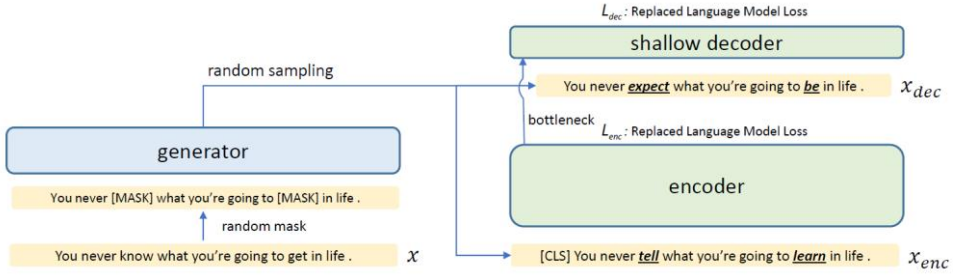

图 6 SimLM 预训练模型架构

如图 6 所示，在预训练阶段有两种随机的操作，一种是随机 mask 掉一系列的词，另一个是通过 ELECTRA 分割的生成器g生成 mask 部分的词。由于随机性这种操作可能会得到相同的句子。根据输入编码器和解码器的不同，上面的操作会采用不同的概率p_{enc}和p_{dec}进行两次：

$$x_{enc} = Sample(g, Mask(x, p_{enc}), x_{dec} = Sample(g, Mask(x, p_{dec})$$

结构中的编码器是多层的 transformer 模型，可以使用 BERT 等模型进行初始化。解码器是 2 层的 transformer 模型。预训练任务采用替换后的掩码语言建模。在预训练结束后丢弃解码器只保留编码器做后续的有监督微调任务。

二、微调阶段

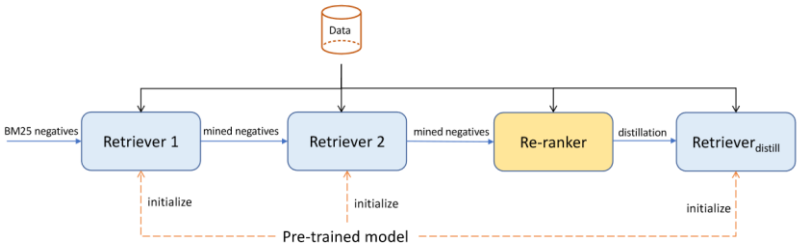

图 7 SimLM 微调模型架构

如图 7 所示，在模型的微调阶段，使用预训练阶段的模型初始化 retriver1 和 retriever2，而对于 re-ranker 采用 ELECTRA 预训练模型进行微调。整个微调阶段比较直观而且不需要联合训练或者定期重建索引。

2.2.3 CoT-MAE

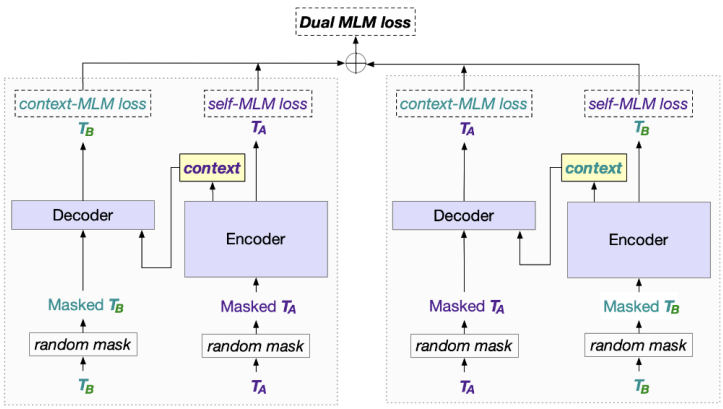

图 8 CoT-MAE 模型架构

CoT-MAE 的设计是为了联合学习文本段内部 token 的语义信息以及两段文本段T_A和T_B之间的语义关系。为此其结构采用了非对称编码器解码器结构。其中编码器采用的是一个较深的网络并有足够的参数，能够学习到较好的文本表征建模能力。解码器采用的是浅层的网络，目的是希望解码尽量依赖于编码器传入的上下文，从而强迫编码器学习一个更好的文本表征。

在预训练损失计算部分，CoT-MAE 分别计算了自监督 MLM 损失以及语义监督 MLM 损失。其中自监督的损失和普通的 MLM 一样，通过将文本段随机 mask 掉 15%，通过没有 mask 的部分预测 mask 的部分，其损失函数为：

$$\mathcal{L}_{smlm}^{A} = -\sum_{t \in m(\mathbf{T_A})} \log p\left(t \mid \mathbf{T_A} \backslash m(\mathbf{T_A})\right)$$

其中$m(T_A)$为A文本段中 mask 掉的部分，$T_A \backslash m(T_A)$表示A文本段中没有 mask 的部分。

语义监督 MLM 则考虑从另一段文本段的没有 mask 的文本以及上一阶段编码器的上下文 embedding 学习 mask 部分的文本。其损失为：

$$\mathcal{L}_{cmlm}^{AB} = -\sum_{t \in m(\mathbf{T_B})} \log p\left(t \mid [h_0^{\text{last}}, \mathbf{T_B} \backslash m(\mathbf{T_B})]\right)$$

其中h_0^{last}是编码器的最后一层隐藏层输出的上下文 embedding，$m(T_B)$为B文本段中 mask 掉的部分，$T_B \backslash m(T_B)$表示B文本段中没有 mask 的部分。

总的损失为考虑 A 和 B 的对偶的形式：

$$\mathcal{L}^{AB} = \mathcal{L}_{smlm}^{A} + \mathcal{L}_{cmlm}^{AB}$$
$$\mathcal{L}^{BA} = \mathcal{L}_{smlm}^{B} + \mathcal{L}_{cmlm}^{BA}$$
$$\mathcal{L} = \mathcal{L}^{AB} + \mathcal{L}^{BA}$$

2.3 实验效果对比

针对前面提出的三类共七种预训练模型，三种后续的微调结构以及 Warmup 训练策略，我们在 NVIDIA Tesla V100 32G * 4 上对这 42 种组合分别进行训练，batch_size 统一设置为 256，得到的训练结果如表 2 所示：

表 2 训练与预测结果（NDCG@10）

模型架构	no warmup			warmup		
	①	②	③	①	②	③
albert-base	0.6658	0.6482	0.3932	0.6630	0.6244	0.4688
bert-base	0.6214	0.6209	0.5508	0.6368	0.6215	0.5163
electra-base	0.7067	**0.7155**	0.4300	0.6911	0.7035	0.3550
simlm-msmarco	0.6672	0.6523	0.5997	**0.6739**	0.6553	0.6049
cocodr-msmarco	0.6432	0.6443	0.6699	0.6537	0.6423	0.6578
cotmae-reranker	0.7605	0.7368	0.6237	**0.7615**	0.7550	0.5803
simlm-reranker	0.7472	0.7368	0.2427	0.7456	0.7222	0.1766

其中，①指后续的模型架构为预训练模型的 pooler_output 经过全连接层后直接输出分数，②指后续的模型架构为预训练模型的 last_hidden_state 经过全连接层后直接输出分数，③指使用 ColBERT 的网络结构。

我们在三种预训练模型上取得的结果分别为：在 Re-ranking 任务上进行训练后的预训练模型为 0.7615；在 msmarco 数据集上经过训练的预训练模型为 0.6739；普通的在其他语料上预训练的模型为 0.7155。

从实验结果可以看出，在 Re-ranking 任务上进行训练后的预训练模型（cotmae_base_msmarco_reranker 和 simlm-msmarco-reranker）的表现最好，NDCG@10 最高可以达到 0.7615。这种结果应该是比较符合常理的，因为模型实际上已经在 Re-ranking 任务上经过了大规模数据的训练，对于我们的小规模数据集几乎不需要怎么进行微调即可取得很好的结果。但是对于在 msmarco 数据集上经过训练的预训练模型（simlm-base-msmarco 和 cocodr-base-msmarco）来说，对于 Re-ranking 任务上的表现甚至还不如普通的在其他语料上预训练的模型（albert-base-v2, bert-base-uncased 和 electra-base-discriminator）。这可能是因为在 msmarco 数据集上经过训练的预训练模型的任务与 Re-ranking 并不匹配，同时我们的训练数据是经过采样后的少量数据，模型并不能学到大规模数据的知识，从而泛化性能并不是很好。

对比三种模型的架构，也可以发现在大多数的情况下，pooler_output 的输出要比 last_hidden_state 的输出更适用于 Re-ranking 任务，但是也并不绝对，在普通的预训练模型的最好结果是在 last_hidden_state 的输出上取得的。而 ColBERT 事实上并不适用于 Re-ranking 任务，ColBERT 更聚焦于 Retrieval 任务，更适合于粗排序，对于 Re-ranking 任务的精排序并不适合。

3 实验步骤

3.1 提交说明

由于文件中包含了原始的预训练模型、训练好的模型和镜像，附件比较大。如果失效，下载地址为：

https://drive.google.com/drive/folders/1jEgrCuCsCVIS1ACE_Brwo0mOm4L1dWNe?usp=sharing

解压 zip 文件至本地目录中：

```
unzip zhangzhao-IR.zip -d zhangzhao-IR
```

附件提供了 Docker 和 python 两种方法运行程序，推荐使用 Docker。如果使用 docker，需要将 IR.tar.gz 解压到 zhangzhao-IR/images 目录下：

```
tar xvzf IR.tar.gz -C zhangzhao-IR/images
```

3.2 文件说明

切换到 zhangzhao-IR 目录下，有如下文件，文件的具体名称和含义如下列所示：

```
.
├── README_submit.md # 说明文件
├── file
│   ├── bert_cat.py # 模型文件 1
│   ├── bert_sequence_classification.py # 模型文件 2
│   ├── colbert.py # 模型文件 3
│   ├── data # 原始数据
│   │   ├── 2019 # 2019 年原始数据
│   │   │   ├── 2019qrels-pass.txt
│   │   │   ├── collection.train.sampled.tsv
│   │   │   ├── msmarco-passagetest2019-43-top1000.tsv
│   │   │   ├── qidpidtriples.train.sampled.tsv
│   │   │   └── queries.train.sampled.tsv
│   │   └── 2020 # 2020 年原始数据
│   │       ├── 2020qrels-pass.txt
│   │       └── msmarco-passagetest2020-54-top1000.tsv
```

```
│   ├── dataset.py # 数据处理文件
│   ├── ensemble_rank.py # 模型集成文件（未使用）
│   ├── ensemble_score.py # 模型集成文件（未使用）
│   ├── log.py # 日志配置文件
│   ├── main.py # 主文件
│   ├── models # 训练好的模型文件
│   │   ├── cotmae_base_msmarco_reranker-bert_sequence_classification
│   │   │   └── best.pt
│   │   ├── electra-base-discriminator-bert_cat
│   │   │   └── best.pt
│   │   └── simlm-base-msmarco-bert_sequence_classification
│   │       └── best.pt
│   ├── only_predict # 仅预测脚本
│   │   ├── run_cotmae_base_msmarco_reranker-
bert_sequence_classification.sh
│   │   ├── run_electra-base-discriminator-bert_cat.sh
│   │   └── run_simlm-base-msmarco-bert_sequence_classification.sh
│   ├── pretrained # 预训练模型文件
│   │   ├── caskcsg
│   │   │   └── cotmae_base_msmarco_reranker
│   │   │       ├── config.json
│   │   │       ├── pytorch_model.bin
│   │   │       ├── special_tokens_map.json
│   │   │       ├── tokenizer_config.json
│   │   │       └── vocab.txt
│   │   ├── google
│   │   │   └── electra-base-discriminator
│   │   │       ├── config.json
│   │   │       ├── pytorch_model.bin
│   │   │       ├── tokenizer.json
│   │   │       ├── tokenizer_config.json
│   │   │       └── vocab.txt
│   │   └── intfloat
│   │       └── simlm-base-msmarco
│   │           ├── config.json
│   │           ├── pytorch_model.bin
│   │           ├── special_tokens_map.json
│   │           ├── tokenizer.json
│   │           ├── tokenizer_config.json
│   │           └── vocab.txt
│   ├── requirements.txt # 依赖库文件
│   ├── result # 运行结果
│   │   ├── log_2022_11_20_11_47_09 # 运行结果日志
│   │   ├── log_2022_11_21_05_01_28 # 运行结果日志
```

```
│   │   ├── log_2022_11_21_05_05_06 # 运行结果日志
│   │   ├── result_2019qrels_2022_11_20_11_47_09 # 在 2019 年数据上进行验证的 TREC 结果文件
│   │   ├── result_2019qrels_2022_11_21_05_01_28 # 在 2019 年数据上进行验证的 TREC 结果文件
│   │   ├── result_2019qrels_2022_11_21_05_05_06 # 在 2019 年数据上进行验证的 TREC 结果文件
│   │   ├── result_2020qrels_2022_11_20_11_47_09 # 在 2020 年数据上进行推理后的 TREC 结果文件
│   │   ├── result_2020qrels_2022_11_21_05_01_28 # 在 2020 年数据上进行推理后的 TREC 结果文件
│   │   └── result_2020qrels_2022_11_21_05_05_06 # 在 2020 年数据上进行推理后的 TREC 结果文件
│   ├── rich_progress.py # 命令行美化配置
│   ├── train_and_predict # 训练+训练后预测脚本
│   │   ├── run_cotmae_base_msmarco_reranker-bert_sequence_classification.sh
│   │   ├── run_electra-base-discriminator-bert_cat.sh
│   │   └── run_simlm-base-msmarco-bert_sequence_classification.sh
│   └── trec_eval-9.0.7.tar.gz # 评测脚本
├── images # Docker 镜像路径
│   └── IR.tar # Docker 镜像
├── install_docker.sh # Docker 安装脚本
├── install_python.sh # Python 依赖库安装脚本
├── predict_docker.sh # Docker 仅预测脚本
├── predict_python.sh # Python 仅预测脚本
├── train_and_predict_docker.sh # Docker 训练+预测脚本
├── train_and_predict_python.sh # Python 训练+预测脚本
```

3.3 训练流程

为了避免将 2019 年的数据和 2020 年的数据进行混用，在最终提交版本的代码中首先将 2019 年的数据与 2020 年的数据分开，保证在训练过程中仅使用 2019 年的数据进行训练和验证，只有在训练完成后预测的时候才去选择 2020 年的数据进行测试。

```
├── 2019 # 2019 年原始数据
│   ├── 2019qrels-pass.txt
│   ├── collection.train.sampled.tsv
│   ├── msmarco-passagetest2019-43-top1000.tsv
│   ├── qidpidtriples.train.sampled.tsv
│   └── queries.train.sampled.tsv
```

```
└── 2020 # 2020年原始数据
    ├── 2020qrels-pass.txt
    └── msmarco-passagetest2020-54-top1000.tsv
```

对于数据的预处理，我们通过 collection.train.sampled.tsv 和 queries.train.sampled.tsv 内部的文本与 ID 的对应关系，将 qidpidtriples.train.sampled.tsv 中每条查询的一个正样本和负样本的 ID 替换成实际的文档，随后通过预训练模型自带的 tokenizer 分词器进行处理，输入到 dataloader 中进行训练和测试。整个过程完全在线进行，不需要提前对原始数据进行处理。

训练的主入口文件是 file/main.py，其中提供了 22 个超参数用于对代码中的可选项进行选择，保证代码运行过程的鲁棒性。

训练脚本如下所示：

```
# Variables
SEED=42
GPU='0 1 2 3'
TRAIN_BATCH=256
MODEL=bert_sequence_classification
BERT='pretrained/caskcsg/cotmae_base_msmarco_reranker'
TIMESTAMP=run3

# 使用2019年的数据进行训练
python main.py \
--train \
--batch ${TRAIN_BATCH} --datetime ${TIMESTAMP} --epoch 20 --gpu ${GPU}
--lr 3e-5 --seed 42 --early_stop 20 \
--data_folder_dir 2019 --train_document collection.train.sampled.tsv --
train_query queries.train.sampled.tsv --qid_pid
qidpidtriples.train.sampled.tsv --test_data_file msmarco-
passagetest2019-43-top1000.tsv --test_result_file 2019qrels-pass.txt \
--save --bert ${BERT} --model ${MODEL} --warmup 0.1
```

脚本的超参数设置，首先保证是训练模式，随后设置了训练的 batch_size 大小、输出文件的时间戳（用于隔离不同的结果文件）、训练轮数、使用的 GPU、学习率、随机种子以及早停轮数（如果在长时间内模型在验证集的效果上没有提升，则认为已经没有再次训练的必要了）。然后设置了训练阶段的数据集，可以看出使用的都是 2019 年的数据集，2020 年的数据并没有参与到训练过程的任何

部分；最后设置了使用的预训练模型、后面的微调策略以及 warmup 策略的学习速率。

设置好脚本后即可通过运行脚本直接训练，训练过程中会同时在控制台与文件中输出日志，从而保证训练结果的可观察性。同时如果添加了--board 选项，会在 tensorboard 中输出关键的指标，如学习率、损失以及目前的 NDCG@10 指标等。

```
2022/11/20 11:50:03 [INFO] main.88 Seed: 42
2022/11/20 11:50:03 [INFO] main.377 Load pretrained/google/electra-base-discriminator Model
2022/11/20 11:50:05 [INFO] main.111 Read test data: data/2019/msmarco-passagetest2019-43-top1000.tsv
2022/11/20 11:50:27 [INFO] main.106 Read train data
2022/11/20 11:50:27 [INFO] main.116 Start Training!
2022/11/20 11:50:32 [INFO] main.123 Load pretrained/google/electra-base-discriminator Tokenizer
2022/11/20 11:52:48 [INFO] main.271 Start evaluate!
2022/11/20 11:53:25 [INFO] main.335 NDCG_10: 0.6766
2022/11/20 11:53:25 [INFO] main.243 Eval Epoch = 1 NDCG_10:0.6766
2022/11/20 11:53:25 [INFO] main.246 Test NDCG_10:0.6766 > max_metric!
2022/11/20 11:53:27 [INFO] main.251 Best Model Saved!
2022/11/20 11:55:05 [INFO] main.271 Start evaluate!
2022/11/20 11:55:42 [INFO] main.335 NDCG_10: 0.6812
2022/11/20 11:55:42 [INFO] main.243 Eval Epoch = 2 NDCG_10:0.6812
2022/11/20 11:55:42 [INFO] main.246 Test NDCG_10:0.6812 > max_metric!
2022/11/20 11:55:44 [INFO] main.251 Best Model Saved!
2022/11/20 11:57:21 [INFO] main.271 Start evaluate!
```

图 9 训练日志示意图

在训练的过程中，每训练一轮就调用评价脚本进行一轮测试，如果测试的结果比上一轮的结果更好，就会保存一个模型到本地，或者将上一个保存的模型替换掉。这样训练结束后最终保存下来的模型是在 2019 年数据上测试的最好的模型。中间并没有 2020 年的数据参与。

3.4 预测流程

预测流程的超参数配置大致如下所示：

```
# Variables
SEED=42
GPU='0 1 2 3'
TEST_BATCH=1024
MODEL=bert_sequence_classification
BERT='pretrained/caskcsg/cotmae_base_msmarco_reranker'
```

```
TIMESTAMP=run3
# 测试 2020 年的数据
python main.py \
--predict \
--batch ${TEST_BATCH}  --datetime ${TIMESTAMP} --gpu ${GPU} \
--data_folder_dir 2020 --test_data_file msmarco-passagetest2020-54-
top1000.tsv --test_result_file 2020qrels-pass.txt \
--load --bert ${BERT} --model ${MODEL}
```

其中的时间戳会去寻找刚刚对应时间戳下训练好的模型，测试的数据均为 2020 年的数据，其余的参数与训练大致相同。测试过程中也会同时在文件中和控制台输出对应的预测信息，并且在最后一行打印测试指标。

```
2022/11/20 12:38:19 [INFO] main.67 Log is ready!
2022/11/20 12:38:19 [INFO] main.68 Namespace(batch=1024, bert='pretrained/google/electra-base-discriminator', board
2022/11/20 12:38:19 [INFO] main.77 GPU: 8,9,10,11
2022/11/20 12:38:19 [INFO] main.88 Seed: 42
2022/11/20 12:38:19 [INFO] main.377 Load pretrained/google/electra-base-discriminator Model
2022/11/20 12:38:21 [INFO] main.111 Read test data: data/2020/msmarco-passagetest2020-54-top1000.tsv
2022/11/20 12:38:48 [INFO] main.271 Start evaluate!
2022/11/20 12:38:53 [INFO] main.281 best.pt Loaded!
2022/11/20 12:39:47 [INFO] main.335 NDCG_10: 0.7155
```

图 10 测试日志示意图

4 参考文献

[1] Yu Y, Xiong C, Sun S, et al. COCO-DR: Combating Distribution Shifts in Zero-Shot Dense Retrieval with Contrastive and Distributionally Robust Learning[J]. arXiv preprint arXiv:2210.15212, 2022.

[2] Khattab O, Zaharia M. Colbert: Efficient and effective passage search via contextualized late interaction over bert[C]//Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval. 2020: 39-48.

[3] Santhanam K, Khattab O, Saad-Falcon J, et al. Colbertv2: Effective and efficient retrieval via lightweight late interaction[J]. arXiv preprint arXiv:2112.01488, 2021.

[4] Wu X, Ma G, Lin M, et al. Contextual mask auto-encoder for dense passage retrieval[J]. arXiv preprint arXiv:2208.07670, 2022.

[5] Wang L, Yang N, Huang X, et al. Simlm: Pre-training with representation bottleneck for dense passage retrieval[J]. arXiv preprint arXiv:2207.02578, 2022.

[6] Craswell N, Mitra B, Yilmaz E, et al. Overview of the TREC 2019 deep learning track[J]. arXiv preprint arXiv:2003.07820, 2020.

[7] Craswell, N., Mitra, B., Yilmaz, E., and Campos, et al. Overview of the TREC 2020 deep learning track[J]. arXiv preprint arXiv: 2102.07662, 2021.

[8] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805.

[9] Clark, K., Luong, M. T., Le, Q. V., & Manning, C. D. (2020). Electra: Pre-training text encoders as discriminators rather than generators. arXiv preprint arXiv:2003.10555.