【RAG】如何炼成强大的向量召回模型

原创 战士金 炼钢AI 2024年10月08日 09:35 北京 标题已修改

和朋友一起写的新书《大模型RAG实战》最近出版啦，这是市面上第一本专门讲解RAG的书籍。关注公众号“炼钢AI”，回复“RAG”获得抽奖链接，抽取1~3位幸运读者送书，ddl：10.13 23:00。

本文在2023年10月首发于我的zhihu帐号：战士金。内容略有修改。

① 前言

对用户提的问题，向量化之后，从向量数据库里召回和用户问题相似的文档片段，是提高大模型回答质量的有效手段（也叫检索增强生成，RAG）。优化向量化模型的效果可进一步提高大模型的回答效果。在通用向量化模型中，baai的bge模型不管是在中文还是英文榜单上，效果都处于遥遥领先的地位。bge1用到的特殊预训练方式、难例挖掘等手段还算是比较常规，bge2文章里的LLM辅助生成软标签、稳定蒸馏等手段就算做的比较深入了。本篇文章里的内容大部分出自于bge的这两篇文章，也会贴上一些自己看过的相关论文以及个人观点。本文主要关注向量化召回模型效果优化的手段，bge原论文里关于他们新提出的训练集、评估基准等内容就不介绍了。

```
1
2 ttps://github.com/FlagOpen/FlagEmbedding/tree/master
3   https://arxiv.org/pdf/2309.07597.pdf
4
5 ttps://github.com/FlagOpen/FlagEmbedding/tree/master/FlagEmbedding/llm_embedder
6   Retrieve Anything To Augment Large Language Models
```

② 带有prompt的向量化召回

在大模型微调这块，有个hard prompt tuning的方式，在进行多任务微调的时候，给不同的任务的input前边都加入固定模式的文字，让模型学会，看到某一段文字之后，就知道要做什么任务了，有助于提高下游不同任务的效果。举一反三，都是语言模型，当然也可以在向量化模型上用这个trick了。即做不同的任务时，分别给不同任务的query和key加上不同的prompt之后在做向量化。我发现的最早的两篇相关文章如下，都是22年的，不知道是偶然撞idea了，还是借鉴了。。

```
1 《One Embedder, Any Task: Instruction-Finetuned Text Embeddings 》
2 《Task-aware Retrieval with Instructions》
```

bge1做向量化召回时候只将召回任务分成两类：对称检索（相似句匹配）和非对称检索（QA匹配）（不明白对称/非对称检索的可以参考我的这篇文章）。如果做是QA匹配，需要

在Q进行向量化时候，加入前缀：“为这个句子生成表示以用于检索相关文章：”。

bge2更是将带有prompt的向量化召回进一步发扬光大，细化成了6大类任务（这块还挺有意思的，有些应用场景之前并没有想到）：

1.知识增强场景（qa）：最直观能想到的场景，我们使用外挂知识库或者说检索增强生成（RAG）给大模型提供额外的知识。query为用户问题，key为包含知识点的文档片段。

2.In-context Learning场景（icl）：few shot场景下，query为问题，key为相关问题的示例，也能提高回答效果。（这个场景挺有意思，有利于刷榜用:)）

3.召回工具场景（tool）：agent场景下，有时候可能需要用到合适的外部工具，query为用户问题，key为工具的文本描述。

4.长对话场景（chat）：多轮对话场景下，query为问题，key为历史对话。

5.对话式搜索（convsearch）：query为用户当前问题和历史对话，key为相关文档片段。

6.长范围语言建模（lrlm）：query为当前当前文本片段，key为之前的文本片段。目标是生成接下来的文本。

各个任务在query和key之前加的prompt如下所示（暂不支持中文）。如果我们想用开源向量化模型在特殊领域数据上微调，当然也可以设计自己的prompt，加载训练数据上。

```
"qa": {
        "query": "Represent this query for retrieving relevant documents: ",
        "key": "Represent this document for retrieval: ",
    },
    "icl": {
        "query": "Convert this example into vector to look for useful example
        "key": "Convert this example into vector for retrieval: ",
    },
    "chat": {
        "query": "Embed this dialogue to find useful historical dialogues: ",
        "key": "Embed this historical dialogue for retrieval: ",
    },
    "lrlm": {
        "query": "Embed this text chunk for finding useful historical chunks:
        "key": "Embed this historical text chunk for retrieval: ",
    },
    "tool": {
        "query": "Transform this user request for fetching helpful tool descr
        "key": "Transform this tool description for retrieval: "
    },
    "convsearch": {
        "query": "Encode this query and context for searching relevant passag
        "key": "Encode this passage for retrieval: ",
    }
```

③ 对比学习与垂直领域微调

对比学习是优化向量化模型的常用训练方法，目的是：优化向量化模型，使其向量化后的文本，相似的在向量空间距离近，不相似的在向量空间距离远。文档召回场景下，做对比学习（有监督）需要三元组（问题，文档正例，文档负例）。文档正例是和问题密切相关的文档片段，文档负例是和问题不相关的文档片段，可以是精挑细选的（难例挖掘出来的，下一小节介绍），也可以是随机出来的。如果是随机出来的话，完全可以用同一个batch里，其他问题的文档正例当作某一个问题的文档负例，如果想要效果好，还需要有比较大的batch size。损失函数是基于批内负样本的交叉熵损失，如下公式所示：

$$\ell_i = -\log \frac{e^{\text{sim}\left(\mathbf{q}_i,\mathbf{d}_i^+\right)/\tau}}{\sum_{j=1}^{N} e^{\text{sim}\left(\mathbf{q}_i,\mathbf{d}_j^+\right)/\tau}}$$

q、d分别表示问题和文档正例对应的向量， τ 为温度系数,sim函数可以是cos相似度或者点积。向量化模型对比学习这块，其实非常值得完整的看看《SimCSE: Simple Contrastive Learning of Sentence Embeddings》这篇经典论文，实验充分，理论扎实。

具体代码实现也非常简单，如下所示。分别将B1个问题，和B2个文档片段通过向量化模型变成向量形式，然后通过矩阵乘积计算每个问题和文档的相似度，最后通过交叉熵损失进行优化。如果文档负例仅来自于同一个batch的其他样本的文档正例，那么B1=B2；如果人工的给每个样本陪k个文档负例（比如可以通过难例挖掘得到），那么B2 =(k+1)*B1。

```
q_reps = self.encode(query) # 问题矩阵 维度(B1, d)
d_reps = self.encode(doc) # 文档矩阵 维度(B2, d)
score = torch.matmul(q_reps, d_reps.transpose(0, 1)) #计算相似度矩阵 维度:(B1, B
scores = scores / self.temperature
target = torch.arange(scores.size(0), device=scores.device, dtype=torch.long)
# 考虑文档负例不仅来自于batch内其他样本的文档正例，也可能人工的给每个样本构造一些文档负
target = target * (p_reps.size(0) // d_reps.size(0))
loss = cross_entropy(scores, target) //交叉熵损失函数
```

bge2论文里，做基于批内负样本的对比学习时同时考虑了多任务问题。之前也介绍了，不同任务加的prompt是不同的，如果把不同任务的样本放到一个batch里，模型训练时候就容易出现偷懒的情况，有时候会根据pormpt的内容来区分正负例，降低任务难度，这是不利于对比学习效果的。因此，可以通过人为的规定，同一个batch里，只能出现同一种任务的样本缓解这个问题。（实际应用场景下，如果任务类别不是非常多的话，最好还是一个任务训练一个模型，毕竟向量化模型也不大，效果会好一些）

在垂直领域应用场景中，亲测构建几千条样本进行对比学习微调就能带来10%左右的召回率提升，性价比非常高。数据可以借助chatgpt进行构造，比如让他根据某段文本内容提出问题，bge项目就自带微调代码，在bge模型的基础上进行微调效果就比较不错了。

④ 难例挖掘

基于批内负采样的对比学习本质是随机选取文档负例，如果能有针对性的，找到和文档正例比较像的文档负例（模型更难区分这些文档负例），加到训练里，是有助于提高对比学习

效果的。就好比我们只有不断的做难题才能更好的提高考试水平。bge进行难例挖掘使用了如下文章的方法《Approximate nearest neighbor negative contrastive learning for dense text retrieval.》，大致思路是在文档向量空间找到和文档正例最相近的文档片段当作文档负例，训练向量化模型。模型更新一段时间后，刷新文档向量，寻找新的文档负例，继续训练模型。还有一些其他的难例挖掘论文，供大家参考。（对比学习难例挖掘的方法图像领域会更多一些？笔者不太了解图像领域）

```
《Contrastive learning with hard negative samples》
《Hard negative mixing for contrastive learning》
《Optimizing dense retrieval model training with hard negatives》
《SimANS: Simple Ambiguous Negatives Sampling for Dense Text Retrieval》
```

除了算法层面，业务层面也可以构造难例，比如文档正例所在的文章里，其他文档片段当作难负例，毕竟至少是属于同一主题的，和随机样本比起来比较难区分。实际应用场景下，如果你的数据比较脏，难例挖掘用处可能不大。

⑤ LLM辅助生成软标签及蒸馏

我们根据用户问题召回的相关文档片段最终是要为LLM回答问题服务的，因此LLM认为召回的文档是否比较好很重要，以下介绍的方法是bge2提出的。对于向量化模型的训练，可以让LLM帮忙生成样本的辅助标签，引导向量化模型训练。辅助标签的生成可用如下公式表示。在已知LLM需要输出的标准答案下，分别将问题和各个文档片段C放入LLM的prompt中，看LLM生成标准答案的概率r大小，当作辅助标签。r越大，表示起对应的文档片段对生成正确答案的贡献越大，也就越重要。但这个打标要求有点太高了吧。。。很多实际应用场景，我们并没法拿到LLM回答的标准答案，同时对每个问题的候选文档片段都计算一个r，开销貌似有点大。。。

$$r_{C|O} = \prod|_{i=1}^{|O|} \mathrm{LLM}(o_i|C, O_{:i-1})$$

利用以上LLM生成的标签以及KL散度（笔者认为论文里这个形式的公式不能叫做KL散度吧。。），对模型进行优化。P为某个问题q对应的候选文档片段p的集合，e表示向量，<.,.>表示相似度操作，w是对所有候选文档p对应的辅助标签值r经过softmax变换后的值。

$$\min. \sum_{\mathcal{P}} -w_i * \log \frac{\exp(\langle e_q, e_p \rangle / \tau)}{\sum_{p' \in \mathcal{P}} \exp(\langle e_q, e_{p'} \rangle / \tau)}.$$

本质是，如果LLM认为某个文档片段越重要，给它的优化权重越大。为了进一步稳定蒸馏效果，还可以对候选文档片段根据r进行排序，只用排名靠后的样本进行优化。

RAG其他相关文章：

```
大模型外挂(向量)知识库：https://zhuanlan.zhihu.com/p/633671394
【RAG】如何更有效的利用召回的文档：https://zhuanlan.zhihu.com/p/651932402
【RAG】大模型辅助召回：https://zhuanlan.zhihu.com/p/653808554
【RAG】召回文档排序策略再思考与实验：https://zhuanlan.zhihu.com/p/656783040
【RAG】基于深度学习的文本切块方法：https://zhuanlan.zhihu.com/p/670604247
```

```
6 【RAG】RAG召回环节评估实践：https://zhuanlan.zhihu.com/p/675560679
```

本人业余时间正在从零预训练一个1B的LLM，使用T级别数据，模型预训练部分已经完成，正在进行微调和评估，欢迎关注：

```
1 https://github.com/zhanshijinwat/Steel-LLM
```

修改于2024年10月08日