一文通透Text Embedding模型：从text2vec、openai-text embedding到m3e、bge

前言

本文已经是今年的第31篇大模型相关的技术文章了，如果说

- 半年之前写博客，更多是出于个人兴趣 + 读者需要
- 那自我司于23年Q3组建LLM项目团队之后，写博客就成了：个人兴趣 + 读者需要 + 项目需要
  如此兼备三者，实在是写博客之幸运矣

我和我司更非常高兴通过博客、课程、内训、项目，与大家共同探讨如何把先进的大模型技术更好、更快的落地到各个行业的业务场景中，赋能千千万万公司的实际业务

而本文一开始是属于：因我司第三项目组「知识库　问答项目」而起的此文《知识库问答LangChain+LLM的二次开发：商用时的典型问题及其改进方案》中的1.2节(*该1.2节初稿来自我司LLM项目团队第三项目组的bingo*)，但为把Text Embedding模型阐述的更为精准、全面，特把那部分的内容抽取出来，不断完善成此文

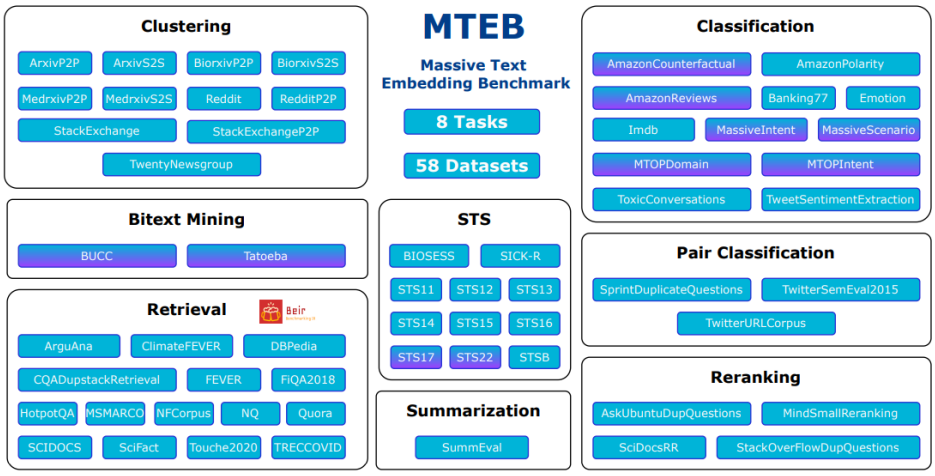
最终尽可能相比网上已有的其他资料都更细致化

第一部分 衡量文本向量表示效果的榜单：MTEB、C-MTEB

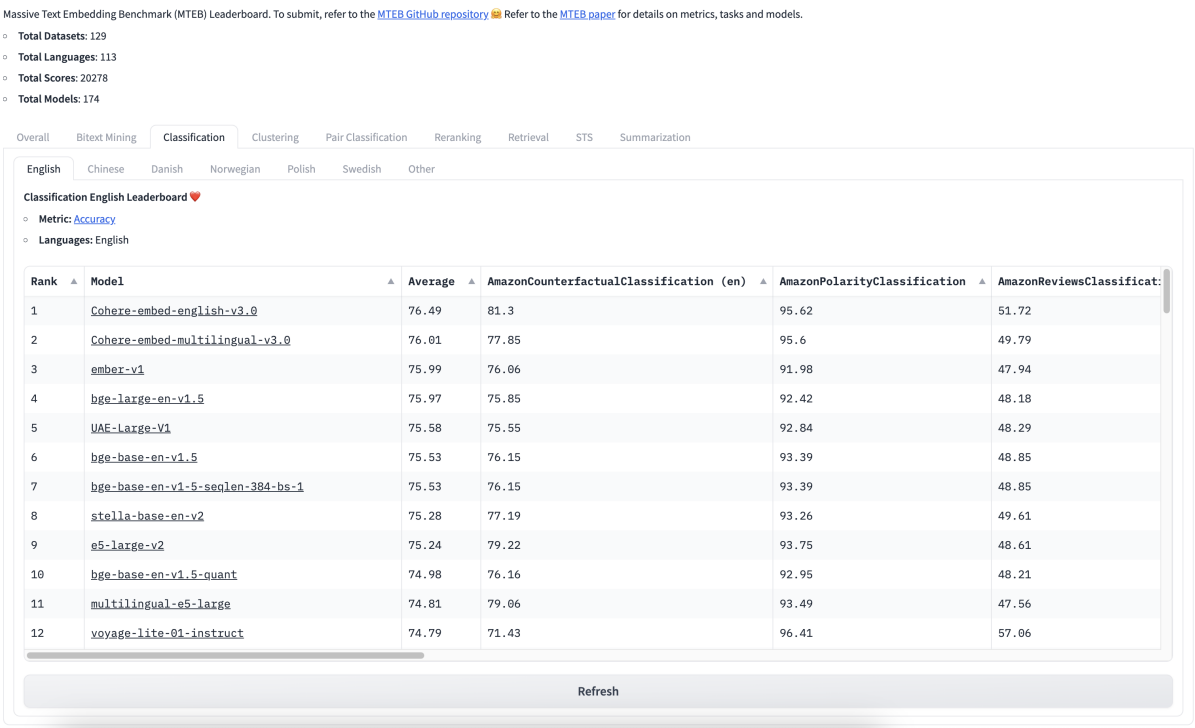
1.1 《MTEB: Massive Text Embedding Benchmark(海量文本嵌入基准)》

判断哪些文本嵌入模型效果较好，通常需要一个 评估指标　来进行比较，《MTEB: Massive Text Embedding Benchmark(海量文本嵌入基准)》就是一个海量文本嵌入模型的评估基准

- 论文地址：https://arxiv.org/abs/2210.07316
  MTEB包含8个语义向量任务，涵盖58个数据集和112种语言。通过在MTEB上对33个模型进行基准测试，建立了迄今为止最全面的文本嵌入基准。我们发现没有特定的文本嵌入方法在所有任务中都占主导地位。这表明该领域尚未集中在一个通用的文本嵌入方法上，并将其扩展到足以在所有嵌入任务上提供最先进的结果
- github地址：https://github.com/embeddings-benchmark/mteb#leaderboard



榜单地址：https://huggingface.co/spaces/mteb/leaderboard



1.2 中文海量文本embedding任务排行榜：C-MTEB

从Chinese Massive Text Embedding Benchmark中可以看到目前最新的针对中文海量文本embedding的各项任务的排行榜，针对不同的任务场景均有单独的排行榜。

任务榜单包括：

- Retrieval
- STS
- PairClassification
- Classification
- Reranking
- Clustering

其中，在本地知识库任务中，主要是根据问题query的embedding表示到向量数据库中检索相似的本地知识文本片段。因此，该场景主要是Retrieval检索任务。检索任务榜单如下：

Model	T2Retrieval	MMarcoRetrieval	DuRetrieval	CovidRetrieval	CmedqaRetrieval	EcomRetrieval	MedicalR
luotuo-bert-medium	58.67	55.31	59.36	55.48	18.04	40.48	29.
text2vec-large-chinese	50.52	45.96	51.87	60.48	15.53	37.58	30.9
text2vec-base-chinese	51.67	44.06	52.23	44.81	15.91	34.59	27.5
m3e-base	73.14	65.45	75.76	66.42	30.33	50.27	42.
m3e-large	72.36	61.06	74.69	61.33	30.73	45.18	48.6
OpenAI(text-embedding-ada-002)	69.14	69.86	71.17	57.21	22.36	44.49	37.9
multilingual-e5-small	71.39	73.17	81.35	72.82	24.38	53.56	44.8
multilingual-e5-base	70.86	76.04	81.64	73.45	27.2	54.17	48.3
multilingual-e5-large	76.11	79.2	85.32	75.51	28.67	54.75	51.4
BAAI/bge-small-zh	77.59	67.56	77.89	68.95	35.18	58.17	49.
BAAI/bge-base-zh	83.35	79.11	86.02	72.07	41.77	63.53	56.6
bge-large-zh-noinstruct	84.39	81.38	84.68	75.07	41.03	65.6	58.2
bge-large-zh	84.82	81.28	86.94	74.06	42.4	66.12	59.3

目前检索任务榜单下效果最好的是bge系列的bge-large-zh模型，langchain-chatchat项目中默认的m3e-base也处于比较靠前的位置

第二部分 OpenAI的text-embedding模型：从ada-002到3-small/3-large

2.1 text-embedding-ada-002

2.1.1 模型简介

text-embedding-ada-002是OpenAI于2022年12月提供的一个embedding模型，但需要调用接口付费使用。其具有如下特点：

- 统一能力：OpenAI通过将五个独立的模型(文本相似性、文本搜索-查询、文本搜索-文档、代码搜索-文本和代码搜索-代码)合并为一个新的模型
  在一系列不同的文本搜索、句子相似性和代码搜索基准中，这个单一的表述比以前的嵌入模型表现得更好
- 上下文：上下文长度为8192，使得它在处理长文档时更加方便
- 嵌入尺寸：只有1536个维度，是davinci-001嵌入尺寸的八分之一，使新的嵌入在处理矢量数据库时更具成本效益

2.1.2 模型使用

以下是OpenAI官方文档中给出的用于文本搜索的代码实例

```
from openai.embeddings_utils import get_embedding, cosine_similarity

def search_reviews(df, product_description, n=3, pprint=True):
  embedding = get_embedding(product_description, model='text-embedding-ada-002')
  df['similarities'] = df.ada_embedding.apply(lambda x: cosine_similarity(x, embedding))
  res = df.sort_values('similarities', ascending=False).head(n)
  return res

res = search_reviews(df, 'delicious beans', n=3)
```

2.3 最新发布的text-embedding-3之small/large的缩短嵌入技术

2.3.1 OpenAI三大嵌入模型的嵌入维度对比

	ada v2	text-embedding-3-small		text-embedding-3-large		
Embedding size	1536	512	1536	256	1024	3072
Average MTEB score	61.0	61.6	62.3	62.0	64.1	64.6

从上图可知，text-embedding-3-small/large这两个新嵌入模型允许开发者通过在 dimensions API 参数中传递嵌入而不丢失其概念表征属性，从而缩短嵌入(即从序列末尾删除一些数字)

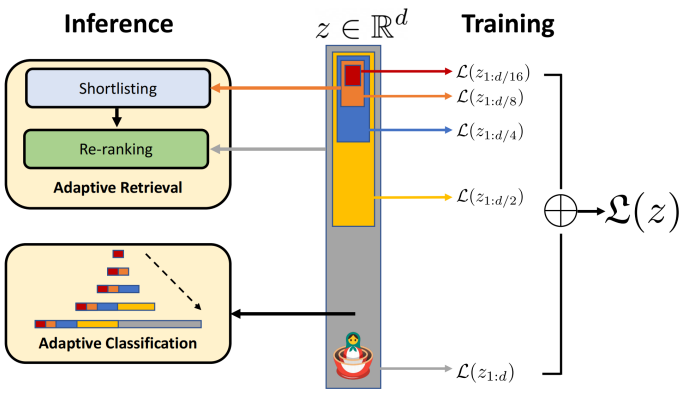
1. 例如在 MTEB 基准上，text-embedding-3-large 可以缩短为 256 的大小， 同时性能仍然优于未缩短的 text-embedding-ada-002 嵌入(大小为 1536)
2. 当然，仍然可以使用最好的嵌入模型 text-embedding-3-large 并指定 dimensions API 参数的值为 1024，使得嵌入维数从 3072 开始缩短，牺牲一些准确度以换取更小的向量大小

2.3.2 Matryoshka Representation Learning

OpenAI 所使用的「缩短嵌入」方法，随后引起了研究者们的广泛注意，最终发现，这种方法和 2022 年 5 月的一篇论文所提出的「Matryoshka Representation Learning」方法是相同的(*MRL 的一作 Aditya Kusupati 也评论道：OpenAI 在 v3 嵌入 API 中默认使用 MRL 用于检索和 RAG！其他模型和服务应该很快就会迎头赶上*)

不同于常规的fix的embedding表征，Matryoshka representation learning提出了一个方法，生成的表征是按照x下标进行重要性排序的，所以在资源受限的环境，可以只使用前面top-k维表征就可以

如下所示，对于$d \in \mathbb{N}$，考虑一组表示尺寸$\mathcal{M} \subset [d]$，对于输入数据X中的数据点x，其的目标是学习一个d维表示向量$z \in \mathbb{R}^d$，对于每一个$m \in \mathcal{M}$，MRL的目标是让前m维的表征向量$z_{1:m} \in \mathbb{R}^m$独立地成为可转移的通用表征向量



再比如，在ImageNet-1K上训练ResNet50，将224×224像素的图像嵌入d=2048表示向量，然后通过线性分类器在$L = 1000$个标签中进行预测$\hat{y}$

1. 对于MRL，选择$\mathcal{M}=\{8,16,\ldots,1024,2048\}$作为嵌套维度
   假设得到了一个带标签的数据集$\mathcal{D}=\{(x_1,y_1),\ldots,(x_N,y_N)\}$，其中$x_i\in\mathcal{X}$是输入点，$y_i\in[L]$是所有$i\in[N]$中$x_i$的标签
2. MRL采用标准的经验风险最小化方法，通过使用独立的线性分类器对每个嵌套维度$m\in\mathcal{M}$进行多类分类损失优化，参数化为$\mathbf{W}^{(m)}\in\mathbb{R}^{L\times m}$
   之后，所有损失分别按各自的重要性$(c_m\geq 0)_{m\in\mathcal{M}}$进行适当缩放后，做最终聚合
   MRL optimizes the multi-class classification loss for each of the nested dimension m ∈M using standard empirical risk minimization using a separatelinear classifier, parameterized by W(m) ∈RL×m .
   All the losses are aggregated after scaling withtheir relative importance (cm ≥0)m∈M respectively

   尽管只对$O(\log(d))$嵌套维度进行优化，MRL仍能产生精确的表示，并对介于所选表示粒度之间的维度进行插值

第三部分 m3e模型

3.1 m3e模型简介

M3E是*Moka Massive Mixed Embedding*的简称，解释一下

- Moka，表示模型由MokaAI训练，开源和评测，训练脚本使用uniem ，评测BenchMark使用 MTEB-zh
- Massive，表示此模型通过千万级(2200w+)的中文句对数据集进行训练
- Mixed，表示此模型支持中英双语的同质文本相似度计算，异质文本检索等功能，未来还会支持代码检索

其有多个版本，分为m3e-small、m3e-base、m3e-large，m3e GitHub地址：GitHub - wangyingdong/m3e-base，其

- 使用in-batch负采样的对比学习的方式在句对数据集进行训练，为了保证in-batch负采样的效果，使用A100来最大化batch-size，并在共计2200W+的句对数据集(*包含中文百科，金融，医疗，法律，新闻，学术等多个领域*)上训练了 1 epoch
- 使用了指令数据集，M3E 使用了300W+的指令微调数据集，这使得 M3E 对文本编码的时候可以遵从指令，这部分的工作主要被启发于 instructor-embedding
- 基础模型，M3E 使用 Roberta 系列模型进行训练，目前提供 small 和 base 两个版本
  此文《知识库问答LangChain+LLM的二次开发：商用时的典型问题及其改进方案》中的langchain-chatchat便默认用的m3e-base

3.1.1 m3e与openai text-embedding-ada-002

以下是m3e的一些重大更新

- 2023.06.08，添加检索任务的评测结果，在 T2Ranking 1W 中文数据集上，m3e-base 在 ndcg@10 上达到了 **0.8004**，超过了 openai-ada-002 的 0.7786
  见下图**s2p ndcg@10**那一列(*其中s2p, 即 sentence to passage ，代表了异质文本之间的嵌入能力，适用任务：文本检索，GPT 记忆模块等*)
- 2023.06.07，添加文本分类任务的评测结果，在 6 种文本分类数据集上，m3e-base 在 accuracy 上达到了 0.6157(*至于m3e-large则是**0.6231***)，超过了 openai-ada-002 的 0.5956
  见下图**s2s ACC**那一列(*其中s2s, 即 sentence to sentence ，代表了同质文本之间的嵌入能力，适用任务：文本相似度，重复问题检测，文本分类等*)

	参数数量	维度	中文	英文	s2s	s2p	s2c	开源	兼容性	s2s Acc	s2p ndcg@10
m3e-small	24M	512	是	否	是	否	否	是	优	0.5834	0.7262
m3e-base	110M	768	是	是	是	是	否	是	优	0.6157	**0.8004**
m3e-large	340M	768	是	否	是	是	否	是	优	**0.6231**	0.7974
text2vec	110M	768	是	否	是	否	否	是	优	0.5755	0.6346
openai-ada-002	未知	1536	是	是	是	是	是	否	优	0.5956	0.7786

此外，m3e团队建议

1. 如果使用场景主要是中文，少量英文的情况，建议使用 m3e 系列的模型
2. 多语言使用场景，并且不介意数据隐私的话，作者团队建议使用 openai text-embedding-ada-002
3. 代码检索场景，推荐使用 openai text-embedding-ada-002
4. 文本检索场景，请使用具备文本检索能力的模型，只在 S2S 上训练的文本嵌入模型，没有办法完成文本检索任务

3.2 m3e模型微调

- 微调脚本：
  m3e是使用uniem脚本进行微调

```
from datasets import load_dataset

from uniem.finetuner import FineTuner

dataset = load_dataset('shibing624/nli_zh', 'STS-B')
# 指定训练的模型为 m3e-small
finetuner = FineTuner.from_pretrained('moka-ai/m3e-small', dataset=dataset)
```

```
finetuner.run(epochs=3)
```

详细教程暂放在「大模型项目开发线上营」中，至于本文后续更新

第四部分 bge模型：采用RetroMAE预训练算法

2023年8月2日，北京智源人工智能研究院发布的中英文语义向量模型BGE(*hf地址：https://huggingface.co/BAAI/bge-large-zh，GitHub地址：https://github.com/FlagOpen/FlagEmbedding/blob/master/README_zh.md*)，以下是BGE的技术亮点

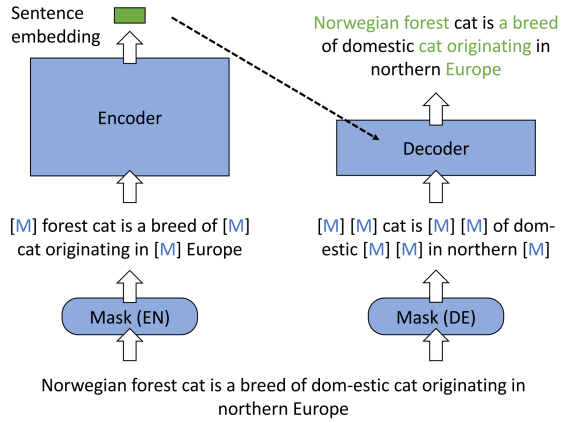
1. 高效预训练和大规模文本微调；
2. 在两个大规模语料集上采用了RetroMAE预训练算法，进一步增强了模型的语义表征能力；
3. 通过负采样和难负样例挖掘，增强了语义向量的判别力；
4. 借鉴Instruction Tuning的策略，增强了在多任务场景下的通用能力

4.1 RetroMAE的预训练步骤

目前主流的语言模型的预训练任务都是token级别的，比如MLM或者Seq2Seq，但是这种训练任务难以让模型获得一个高质量的基于句子级别的句向量，这限制了语言模型在检索任务上的潜力。针对这个弊端，目前有两者针对检索模型的预训练策略

- 第一种是self-contrastive learning，这种方式往往受限于数据增强的质量，并且需要采用非常庞大数量的的负样本
- 另一种基于anto-encoding，一种自重建方法，不受数据增强跟负样本采样策略的影响，基于这种方法的模型性能好坏有两个关键因素
  其一是重建任务必须要对编码质量有足够的要求，其二是训练数据需要被充分利用到

基于此，研究人员提出了RetraoMAE(*RetroMAE论文：https://arxiv.org/abs/2205.12035*)，它包括两个模块，其一是一个类似于BERT的编码器，用于生成句向量，其二是一个一层transformer的解码器，用于重建句子，如下图所示



4.1.1 编码Encoding

所谓编码，即Mask(EN)掉一小部分token然后通过BERT编码得到句子嵌入*sentence embedding*h_x，具体步骤如下

1. 给定一个句子输入X：*Norwegian forest cat is a breed of dom-estic cat originating in northern Europe*
2. 随机Mask(EN)掉其中一小部分token后得到X_{enc}：*[M] forest cat is a breed of [M] cat originating in [M] Europe*
   这里通常会采用一定的mask比例(15%～30%)，从而能保留句子原本大部分的信息
3. 然后利用类似BERT的编码器$\Phi^{enc}(\cdot)$对其进行编码，得到对应的的句子嵌入$\mathbf{h}_{\tilde{x}}$「*一般将[CLS]位置最后一层的隐状态作为句子嵌入*」，如下公式所示

$$\mathbf{h}_{\tilde{x}} \leftarrow \Phi_{enc}\left(\tilde{X}_{enc}\right)$$

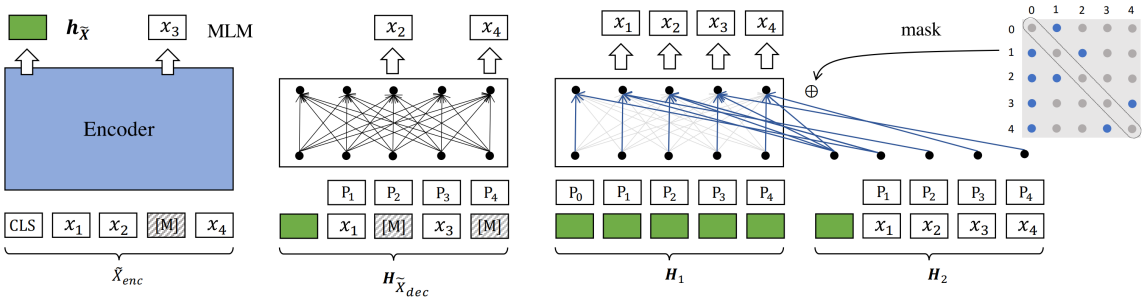
We apply a BERT like encoder with 12 layers and768 hidden-dimensions, which helps to capture thein-depth semantics of the sentence. Following the common practice, we select the [CLS] token's finalhidden state as the sentence embedding.

4.1.2 解码Decoding

所谓解码，即Mask(DE)很大一部分token然后结合句子嵌入*sentence embedding*$\mathbf{h}_{\tilde{x}}$，让解码器重构原始句子

具体而言，即是联合以下两个部分，好让解码器在该两部分的基础上重构原始句子

- 利用Mask(DE)后的文本输入$\tilde{X}_{dec}$：*[MI [M] cat is MI [M} of dom-estic [M] [M] in northern [M]*
  (*这里采取了比encoder部分更加激进的mask比例，比如50%～70%*)
- 与上一节encoder生成的句子嵌入(*sentence embedding*)
  论文中对这一步骤的英文阐述是：The masked input is joined with the sentence embedding, based on which the original sentence is reconstructed by the decoder.



(A) Encoding (B) Decoding (C) Enhanced decoding

> *有个细节是，这里的Mask(DE)输入带上位置嵌入了，即句子嵌入$\mathbf{h}_{\tilde{x}}$和带有位置的掩码输入被组合成以下序列*
>
> $$\mathbf{H}_{\tilde{X}_{dec}} \leftarrow [\mathbf{h}_{\tilde{x}}, \mathbf{e}_{x_1} + \mathbf{p}_1, \ldots, \mathbf{e}_{x_N} + \mathbf{p}_N]$$
>
> *其中，e_{x_i}表示x_i的嵌入，在x_i的基础上增加了一个额外的位置嵌入p_i*
>
> ---
>
> *接下来，通过优化以下目标，学习解码器Φ_{dec}来重构原始句子X*
>
> $$\mathcal{L}_{dec} = \sum_{x_i \in \text{ masked}} \text{CE}\left(x_i \mid \Phi_{\text{dec}}\left(\mathbf{H}_{\tilde{X}_{dec}}\right)\right)$$
>
> 其中，CE是交叉熵损失

由于在解码器部分采用了极其简单的网络结构跟非常激进的mask比例，从而使得解码任务变得极具挑战性，迫使encoder去生成高质量的句向量才能最终准确地完成原文本重建

4.1.3 增强解码Enhanced Decoding

前面提及的解码策略有一种缺陷，就是训练信号只来源于被mask掉的token，而且每个mask掉的token都是基于同一个上下文重建的。于是研究人员提出了一种新的解码方法：Enhanced Decoding，具体做法如下

- a) 首先生成两个不同的输入流H1(query)跟H2(context)

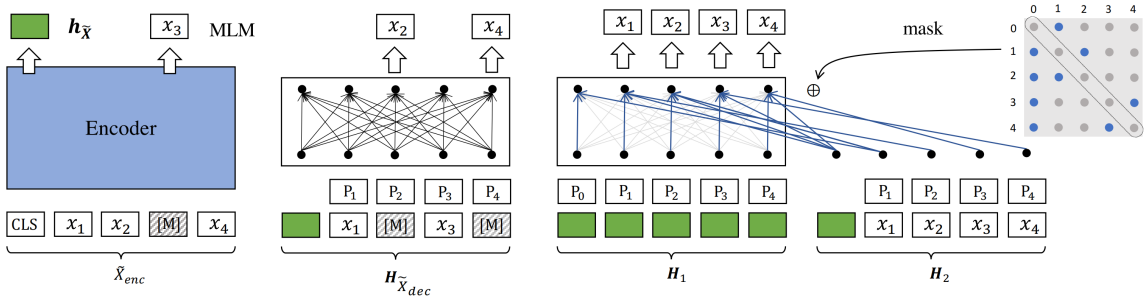
$$\mathbf{H}_1 \leftarrow [\mathbf{h}_{\tilde{X}} + \mathbf{p}_0, ..., \mathbf{h}_{\tilde{X}} + \mathbf{p}_N],$$
$$\mathbf{H}_2 \leftarrow [\mathbf{h}_{\tilde{X}}, \mathbf{e}_{x_1} + \mathbf{p}_1, ..., \mathbf{e}_{x_N} + \mathbf{p}_N].$$

  其中$\mathbf{h}_{\tilde{X}}$是句子嵌入，e_{x_i}是标记嵌入(在这个地方没有标记被掩码)，p_i是位置嵌入
  相当于
  H_1是sentence embedding + Position embedding
  H_2为sentence embedding和token embedding + position embedding

- b) 通过attention机制得到新的输出A

$$\mathbf{Q} = \mathbf{H}_1\mathbf{W}^Q, \mathbf{K} = \mathbf{H}_2\mathbf{W}^K, \mathbf{V} = \mathbf{H}_2\mathbf{W}^V;$$
$$\mathbf{M}_{ij} = \begin{cases} 0, & \text{can be attended,} \\ -\infty, & \text{masked;} \end{cases}$$
$$\mathbf{A} = \text{softmax}(\frac{\mathbf{Q}^T\mathbf{K}}{\sqrt{d}} + \mathbf{M})\mathbf{V}.$$

  这里的M是一个mask矩阵，第i个token所能看得到的其他token是通过抽样的方式决定的(当然要确保看不到自身token，而且都要看得见第一个token，也就是encoder所产出CLS句向量的信息)

> *其中一个和常规decoder不一样的地方是，H1作为Q，H2作为KV*
>
> 
>
>
> (A) Encoding (B) Decoding (C) Enhanced decoding
>
> *H1中的每个token embedding去H2中查找比较重要的上下文：包括**H2被采样到的 token，以及初始token embedding**都能看到[这里的初始embedding就是sentence embedding]，至于对角线上的因代表的各自自身，故看不到*
>
> ---
>
> *为方便大家更好、更快的理解，我再举个例子，比如：*

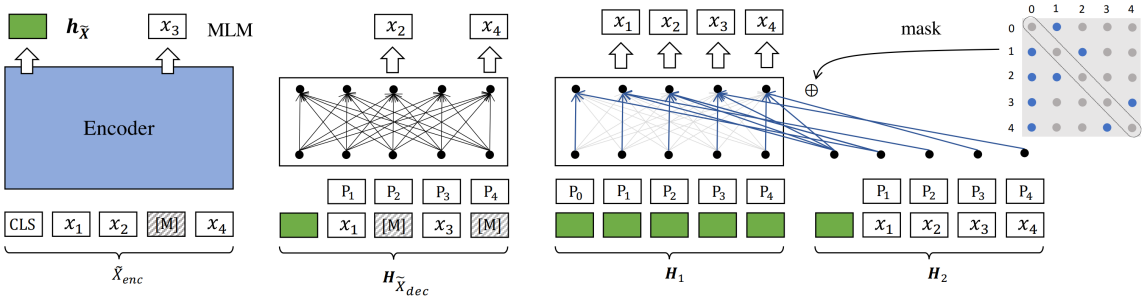
> *P0上的x_0能看见P1上的x_1 (~~x0~~、x1)*
> *P1上的x_1能看见P0、P2上的x_0、x_2 (x0、~~x1~~、x2)*
> *P2上的x_2能看见P0、P1上的x_0、x_1 (x0、x1、~~x2~~)*
> *P3上的x_3能看见P0、P4上的x_0、x_4 (x0、_ 、_、~~x3~~、x4)*
> *P4上的x_4能看见P0、P3上的x_0、x_3 (x0、_ 、_、x3、~~x4~~)*
> *之后，每个token xi基于对矩阵M的第i行可见的上下文进行重构(each token xi is recon-structed based on the context which are visible tothe i-th row of matrix M)，该矩阵即如下所示*
>
> $$\mathbf{M}_{ij} = \begin{cases} 0, & x_j \in s\left(X_{\neq i}\right), \text{ or } j_{|i\neq 0} = 0 \\ -\infty, & \text{otherwise.} \end{cases}$$
>
> 主对角线位置填充为-∞(*因为其代表自身，故不可见*)，可见上下文的位置填充为0，代表可见

- c)最终利用attention后的输出A跟H1一起过FNN(即resnet)去重建原文本，这里重建的目标不仅仅是被mask掉的token，而是全部token

$$\mathcal{L}_{dec} = \sum_{x_i \in X} \mathrm{CE}\left(x_i \mid \mathbf{A}, \mathbf{H}_1\right)$$

最终RetroMAE的损失由两部分相加得到，其一是encoder部分的MLM损失，其二是deocder部分自重建的交叉熵损失

最后，再总结一下RetroMAE 预训练步骤



1. (A)编码阶段：将输入进行一定比例的mask操作，并编码为句子嵌入(绿色矩形)
   (A) Encoding: the input is moderately masked and encoded as the sentence embedding (the green rectangle)
2. (B)解码阶段：对输入使用很高比例的mask操作，并与句子嵌入连接以恢复被mask的部分(阴影符号)
   (B) Decoding: the input is aggressively masked, and joined with the sentence embedding to reconstruct the masked tokens (the shadowed tokens).
3. (C)增强编码阶段：基于每行的句子嵌入和可见上下文来重建所有输入符号；主对角线位置填充为-∞(*灰色，因为其代表自身，故不可见*)，可见上下文的位置填充为0(*蓝色*)
   (C) Enhanced encoding: all input tokens are reconstructed based on the sentence embedding and the visible context in each row (defined in Eq. 7); the main diagonal positions are filled with −∞ (grey), and positions for the visible context are filled with 0 (blue).

4.2 bge模型的微调

- 微调脚本：https://github.com/FlagOpen/FlagEmbedding/tree/master/examples/finetune
- 数据格式

```
{"query": str, "pos": List[str], "neg":List[str]}
```

- 难负样本挖掘
  难负样本是一种广泛使用的提高句子嵌入质量的方法。可以按照以下方法挖掘难负样本

```
python -m FlagEmbedding.baai_general_embedding.finetune.hn_mine \
--model_name_or_path BAAI/bge-base-en-v1.5 \
--input_file toy_finetune_data.jsonl \
--output_file toy_finetune_data_minedHN.jsonl \
--range_for_sampling 2-200 \
--use_gpu_for_searching
```

- 训练

```
python -m FlagEmbedding.baai_general_embedding.finetune.hn_mine \
--model_name_or_path BAAI/bge-base-en-v1.5 \
--input_file toy_finetune_data.jsonl \
--output_file toy_finetune_data_minedHN.jsonl \
--range_for_sampling 2-200 \
--use_gpu_for_searching
```

4.3 新一代通用语义向量模型BGE-M3

4.3.1 BGE-M3：多语言、多粒度、多功能

近日，智源发布了BGE家族新成员——通用语义向量模型BGE-M3(*其开源仓库及技术报告、模型链接*)

1. 多语言Multi-Linguality
   支持超过100种语言，具备领先的多语言、跨语言检索能力

   具体而言，BGE-M3训练集包含100+种以上语言，既包含每种语言内部的语义匹配任务(Language X to Language X)，又包含不同语言之间的语义匹配任务(Language X to Language Y)
   丰富且优质的训练数据帮助BGE-M3建立了出色的多语言检索(Multi-Lingual Retrieval)与跨语言检索能力(Cross-Lingual Retrieval)
2. 多粒度Multi-Granularity
   全面且高质量地支撑“句子”、“段落”、“篇章”、“文档”等不同粒度的输入文本，最大输入长度为 8192

   具体而言，BGE-M3目前可以处理最大长度为8192 的输入文本，极大地满足了社区对于长文档检索的需求。在训练BGE-M3时，智源研究员在现有长文本检索数据集的基础之上，通过模型合成的方式获取了大量文本长度分布多样化的训练数据
   与此同时，BGE-M3通过改进分批(batch)与缓存(cache)策略，使得训练过程具备足够高的吞吐量与负样本规模，从而确保了训练结果的质量。基于数据与算法双层面的优化处理，BGE-M3得以高质量的支持“句子”、“段落”、“篇章”、“文档”等不同粒度的输入文本
3. 多功能Multi-Functionality
   一站式集成了稠密检索、稀疏检索、多向量检索三种检索功能

   具体而言，不同于传统的语义向量模型，BGE-M3既可以借助特殊token [CLS]的输出向量用以来完成稠密检索(Dense Retrieval)任务
   又可以利用其他一般性token的输出向量用以支持稀疏检索(Sparse Retrieval)与多向量检索(Multi-vector Retrieval)

   三种检索功能的高度集成使得BGE-M3可以一站式服务不同的现实场景，如语义搜索、关键字搜索、重排序
   同时，无需使用多个模型进行多个推理，BGE-M3一次推理就可以得到多个不同模式的输出，无需额外开销，并能高效支持混合检索，联合三种检索模式可获得更加精准的检索结果

4.3.2 训练数据

在训练数据上，包括

1. 未标记语料库的弱监督数据
   通过提取Wikipedia、S2ORC、xP3、mC4和CC-News等各种多语言语料库中的丰富语义结构（如标题-正文、标题-摘要、指令-输出等）来进行整理
   为了学习用于跨语言语义匹配的统一嵌入空间，从两个翻译数据集MTP、NLLB中进行处理，总共收集了194种语言的12亿个文本对和2655个跨语言对应
2. 标记语料库的微调数据
   对于英语，包括八个数据集，包括HotpotQA、TriviaQA、NQ、MSMARCO、COLIEE、PubMedQA以及SimCSE收集的NLI数据
   对于中文，包括七个数据集，包括DuReader、mMARCO-ZH、T2-Ranking、LawGPT1、CmedQAv2和LeCaRDv2
   对于其他语言，利用来自MIRACL的训练数据
3. 合成的多语言微调数据
   另外，为了缓解长文档检索任务的性能，通过生成合成数据方式进行处理，并引入额外的多语言微调数据(记为MultiLongDoc)
   具体实现时，从Wiki和MC4数据集中抽取长篇文章，并从中随机选择段落。然后，使用GPT-3.5根据这些段落生成问题，生成的问题和抽样文章构成微调数据的新文本对

   这里采用的prompt为：

   > “You are a curious AI assistant, please generate one specific and valuable question based on the following text.
   > The generated question should revolve around the core content of this text, and avoid using pronouns (e.g., ”this”). Note that you should generate only one question, without including additional content:”

最终得到的数据分布如下：

Language	Source	#train	#dev	#test	#cropus	Avg. Length of Docs
ar	Wikipedia	1,817	200	200	7,607	9,428
de	Wikipedia, mC4	1,847	200	200	10,000	9,039
en	Wikipedia	104,090	200	800	200,000	3,308
es	Wikipedia, mC4	2,254	200	200	9,551	8,771
fr	Wikipedia	1,608	200	200	10,000	9,659
hi	Wikipedia	1,618	200	200	3,806	5,555
it	Wikipedia	2,151	200	200	10,000	9,195
ja	Wikipedia	2,262	200	200	10,000	9,297
ko	Wikipedia	2,198	200	200	6,176	7,832
pt	Wikipedia	1,845	200	200	6,569	7,922
ru	Wikipedia	1,864	200	200	10,000	9,723
th	mC4	1,970	200	200	10,000	8,089
zh	Wikipedia, Wudao	100,834	200	800	200,000	4,249
Summary	Wikipedia, Wudao, mC4	226,358	2,600	3,800	493,709	4,737

4.3.3 混合检索与模型训练

如BGE模型一致，BGE-M3模型训练分为三个阶段：

1. RetroMAE预训练，在105种语言的网页数据和wiki数据上进行，提供一个可以支持8192长度和面向表示任务的基座模型
2. 无监督对比学习，在194种单语言和1390种翻译对数据共1.1B的文本对上进行的大规模对比学习

3. 多检索方式统一优化，在高质量多样化的数据上进行多功能检索优化，使模型具备多种检索能力

4.3.3.1 混合检索

> 从形式上看，在给定一个x表示的查询q时，从语料库D_y中检索出以y表示的文档d：$d_y \leftarrow f_n(q_x, D_y)$
> 其中
> - $f_n()$属于密集检索、稀疏/词汇检索或多维检索中的任何一种函数
> - y可以是另一种语言，也可以是与x相同的语言

M3-Embedding统一了嵌入模型的所有三种常见检索功能，即稠密检索、词性(稀疏)检索和多向量检索

- **sdense，密集检索**
  可以根据文本编码器将输入查询q转换成隐藏状态H_q，继而
  → 使用特殊标记"[CLS]"的归一化隐藏状态来表示查询q的嵌入：$eq = norm(H_q[0])$
  → 使用$ep = norm(H_p[0])$来表示段落p的嵌入
  故，查询和段落之间的相关性得分是通过两个嵌入式eq和ep之间的内积来衡量：$\text{sdense} \leftarrow \langle \text{ep}, \text{eq} \rangle$
- **slex，在词汇检索上，输出嵌入式也用于估算每个词条的重要性**
  而估算词的重要程度，可以方便词汇检索
  具体而言，对于查询中的每个术语t（一个术语对应一个token），术语权重的计算公式为如下：
  $w_{q_t} \leftarrow \text{Relu}\left(\mathbf{W}_{lex}^T \mathbf{H}_{\mathbf{q}}[i]\right)$
  其中$\mathbf{W}_{lex} \in \mathcal{R}^{d \times 1}$是将隐藏状态映射到浮点数的矩阵

  如果一个术语t在查询中出现多次，只保留其最大权重，用同样的方法计算段落中每个术语的权重
  在估算术语权重的基础上，**查询和段落之间的相关性得分**由查询和段落中共存术语(*表示为$q \cap p$，也就是交集*)的联合重要性计算得出(*Based on the estimation term weights, **the relevance score between query and passage** is computed by the joint importance of the co-existed terms (denoted as q ∩ p) with in the query and passage*)：
  $s_{lex} \leftarrow \sum_{t \in q \cap p} (w_{q_t} * w_{p_t})$
- **smul，在多向量检索上，作为密集检索的扩展**
  利用整个输出嵌入来表示查询/和段落，并根据Col-Bert使用后期交互来计算细粒度相关性得分

最后，由于嵌入模型的多功能性，检索过程可以采用混合过程

1. 首先，候选结果可以由每种方法单独检索(多向量方法由于成本高，可以免去这一步骤)
2. 然后，根据综合相关性得分对最终检索结果重新排序：**srank←sdense+slex+smul**

4.3.3.2 模型训练

嵌入模型经过训练，可以区分正样本和负样本。对于每一种检索方法，它都有望为查询的正样本分配一个更高的分数，因此，进行训练过程是为了最小化In-foNCE损失，如下表示

$$\mathcal{L} = -\log \frac{\exp(s(q,p^*)/\tau)}{\sum_{p \in \{p^*, P'\}} \exp(s(q,p)/\tau)}$$

其中

- p^*和P'分别代表查询q的正样本和负样本
- s(-)是{sdense(-)、slex(-)、smul(-)} 中的任意一个函数，不过，不同检索方法的预测结果可以整合为一个更准确的相关性得分，因为它们具有异质性。在最简单的形式中，整合可以是不同预测得分的总和： sinter ←sdense+slex+smul

在知识蒸馏阶段，**以总得分sinter为教师模型监督信号，其中的任何一个得分作为学生模型**

将每种检索方法的损失函数修改为：

$$\mathcal{L}'_* \leftarrow p\left(s_{\text{inter}}\right) * \log p\left(s_*\right)$$

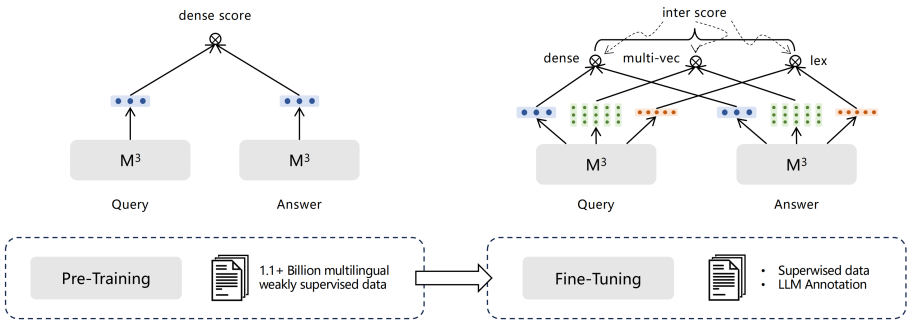
进一步对修正后的损失函数进行积分和归一化处理：

$$\mathcal{L}' \leftarrow \left(\mathcal{L}'_{\text{dense}} + \mathcal{L}'_{\text{lex}} + \mathcal{L}'_{\text{mul}}\right)/3$$

最后使用线性组合推导出自我知识蒸馏的最终损失函数

$$\mathcal{L}_{\text{final}} \leftarrow \mathcal{L} + \mathcal{L}'$$

> 整个训练过程是一个两阶段的工作流程，如下所示：

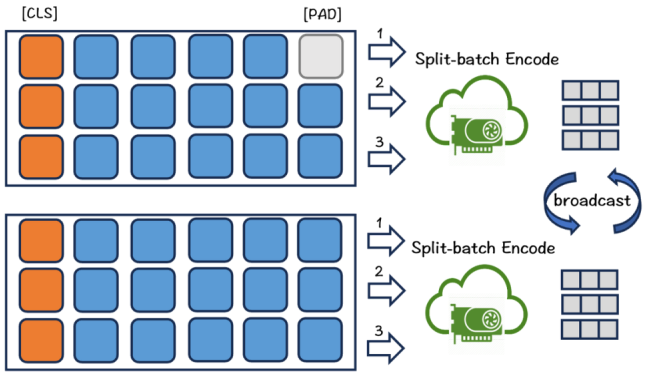


1. 首先，使用弱监督数据对文本编码器进行预训练，其中只有密集检索是以对比学习的基本形式进行训练的
2. 其次，自我知识蒸馏应用于第二阶段，在此阶段对嵌入模型进行微调，以建立三种检索功能。在这一阶段中，将使用标注数据和合成数据，并根据ANCE方法为每个查询引入硬负样本

4.3.4 三大关键技术：自学习蒸馏、训练效率优化、长文本优化等

- **自学习蒸馏**
  人类可以利用多种不同的方式计算结果，矫正误差。模型也可以，通过联合多种检索方式的输出，可以取得比单检索模式更好的效果
  因此，BGE-M3使用了一种自激励蒸馏方法来提高检索性能。具体来说，合并三种检索模式的输出，得到新的文本相似度分数，将其作为激励信号，让各单模式学习该信号，以提高单检索模式的效果
- **训练效率优化**
  由于引入了大量文本长度差异极大的训练数据，常规的对比学习训练方法的效率非常低下
  一方面，长文本会消耗相当多的显存，大大的限制了训练时的batch size
  其次，短文本不得不填充至更长的长度以对齐同一批次的长文本，这样就引入了大量无意义的计算
  此外，训练数据的长度差异容易使得不同GPU之间的计算负荷分布不均并引发相互等待，造成不必要的训练延时

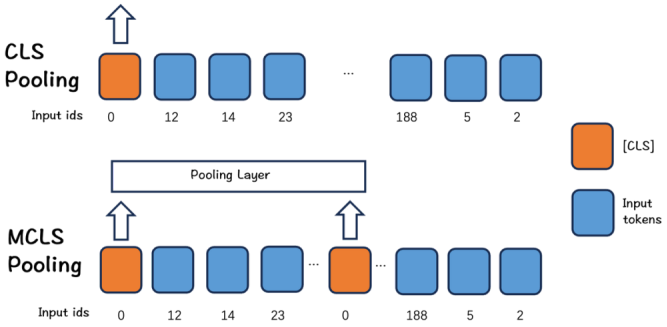
  为了解决这些问题，我们优化了训练流程，如下图所示



→ 首先，根据长度对文本数据进行分组，并从每组中采样数据，确保一个batch内文本长度相对相似，从而减少填充。同时，数据程序会首先从组内采样足够的数据，然后分配给各机器，保证不同机器的计算开销尽可能相近
→ 为了减少文本建模时的显存消耗，我们将一批数据分成多个小批。对于每个小批，我们利用模型编码文本，**收集输出的向量同时丢弃所有前向传播中的中间状态**，最后汇总向量计算损失。通过这种方式，可以显著增加训练的batch size(*且当建模长文本或者向量模型很大时，都可以采用此来扩大batch size*)

- **长文本优化**
  目前，开源社区缺少用于文档级检索的开源数据集。为此，我们借助大语言模型生产了一份包括13种语言的长文档检索数据。另外，由于缺乏长文本数据或计算资源，实际情况下长文本微调不一定可以进行

  在这种情况下，我们提出了一种简单而有效的方法：MCLS(Multiple CLS)来增强模型的能力，而无需对长文本进行微调，如下图所示



MCLS方法旨在利用多个CLS token来联合捕获长文本的语义
具体来说，我们为每个固定数量的token插入一个cls token，每个cls token可以从相邻的token获取语义信息。最后，通过对所有cls token的最后隐藏状态求平均值来获得最终的文本嵌入(*In MCLS,we insert a cls token for every fixed number of tokens, and the final text embedding is obtained by averaging the last hidden states of all cls tokens.*)