**RAG技术笔记**

1. **embedding**

嵌入模型本质上可以看作是一个查找表，它将离散的输入（如单词索引）映射到连续的嵌入向量。这个查找表的权重矩阵是根据训练数据优化的。严格来说，嵌入层并不是神经网络中的典型“层”，因为它没有激活函数和非线性操作，但它确实具有参数（权重），这些参数通过训练进行优化。

详细步骤：

1. **文本预处理**

首先，将输入句子分成单词或子词。这里以单词为例：

sentence = "I like deep learning"

words = sentence.split()

得到的 words 是 ['I', 'like', 'deep', 'learning']。

1. **索引化**

然后，将这些单词转换为词汇表中的索引。假设词汇表如下：

vocab = {'I': 0, 'like': 1, 'deep': 2, 'learning': 3}

word\_indices = [vocab[word] for word in words]

得到的 word\_indices 是 [0, 1, 2, 3]。

1. **嵌入层处理**

接下来，将这些索引输入嵌入层，生成对应的嵌入向量。

1. **聚合操作（可选）**

在某些任务中，可能需要将生成的嵌入向量聚合成一个句子的表示。

# 平均池化

sentence\_embedding = embeddings.mean(dim=0)

print(sentence\_embedding)

1. **chunk、node、index**
2. **Chunk（块）**

Chunk 是指将文档拆分成更小的、可管理的部分。每个块包含一定数量的文本，例如一个段落或几句话。这些块是文档在处理过程中的基本单元，便于系统在检索时快速定位相关信息。

1. **Node（节点）**

Node 是文档的逻辑单元，可以视为文档结构中的一个节点。在分层文档结构中，节点可能代表不同层级的内容，例如章节、节或段落。每个节点包含一个或多个块，并且节点之间可能存在父子关系或相邻关系。

1. **Index（索引）**

Index 是对文档内容进行组织和加速检索的结构。索引可以包含每个块的元数据、关键词、主题等信息，使得系统在检索时能够快速找到包含相关信息的块。常见的索引技术包括倒排索引和基于向量的索引。

1. **(4)Split Strategy（拆分策略）**

Split Strategy 是指将文档拆分成块的具体方法和规则。拆分策略会影响文档块的大小和数量，常见的拆分策略包括：

1. 固定长度拆分：将文档按固定的字数或句子数进行拆分。例如，每200个字或每5句话作为一个块。
2. 基于段落拆分：按照自然段落进行拆分，每个段落作为一个块。
3. 基于语义拆分：利用自然语言处理技术，根据语义边界（如句子之间的逻辑关系或主题变化）进行拆分，确保每个块在语义上尽量独立和完整。
4. 混合策略：结合上述方法，根据具体需求和文档特征进行混合拆分。例如，先按段落拆分，再对过长的段落进行进一步拆分。
5. **retriever、query**

**Retriever（检索器）**

Retriever 是RAG系统中的核心组件之一，负责从大型文档库或知识库中检索与输入查询相关的文档片段（chunks）。Retriever的性能和准确性直接影响到系统的整体效果。常见的retriever类型包括：

1. 稀疏向量检索器：如BM25，基于词频-逆文档频率（TF-IDF）和词条匹配来检索相关文档。适用于需要高效率且对文本长度不敏感的场景。
2. 密集向量检索器：如DPR（Dense Passage Retrieval），使用神经网络将文档和查询编码为向量，通过向量相似性（如内积）进行检索。适用于语义匹配和处理大规模文档库。
3. 混合检索器：结合稀疏和密集向量检索的优势，通过综合两种方法的结果来提高检索准确性。

**Query（查询）**

Query 是用户输入的查询字符串，描述了用户想要获取的信息。在RAG系统中，query通常是自然语言文本。query的处理和优化对于检索结果的质量至关重要。以下是query在RAG系统中的处理过程：

1. 预处理：对输入的query进行基本的预处理，包括分词、去停用词、词干提取等，以提升检索器的效率和准确性。
2. 查询扩展：通过添加同义词、相关词或上下文信息来扩展query，以捕捉更多潜在的相关文档片段。
3. 向量化：在使用密集向量检索器时，将query转换为向量表示，以便与文档片段的向量进行相似性计算。
4. 检索和排序：使用retriever从文档库中检索出与query相关的文档片段，并根据相关性进行排序，返回最相关的片段供生成器使用。
5. **llm、prompt**

**LLM（大语言模型）**

LLM，即大语言模型，是RAG系统中的生成组件，负责根据检索到的文档片段生成最终的回答。大语言模型通常是基于深度学习的神经网络模型，如GPT-3、GPT-4、BERT等。这些模型经过大规模文本数据的训练，具有强大的自然语言理解和生成能力。

在RAG系统中，LLM的作用包括：

1. 回答生成：根据用户的查询和检索到的相关文档片段，生成连贯且准确的回答。
2. 上下文理解：能够理解用户查询的上下文，结合文档片段的信息，生成符合语境的回答。
3. 信息整合：将多个文档片段中的信息整合起来，提供全面且有逻辑的回答。

**Prompt（提示）**

Prompt 是输入到LLM的文本，包含用户的查询和检索到的文档片段。prompt在RAG系统中起到引导LLM生成回答的作用。设计有效的prompt对生成高质量的回答至关重要。prompt环节的主要步骤包括：

1. 构建prompt：将用户的查询和检索到的文档片段组合成一个完整的文本输入。这通常包括将查询放在开头，后面附上相关文档片段。
2. 提示优化：为了提高生成质量，可以对prompt进行优化，包括调整文档片段的顺序、添加引导性的文本等。
3. 输入LLM：将构建好的prompt输入到LLM，等待模型生成回答。

