CH08. 임베딩(Embedding)

Date	@2026년 1월 4일
Category	GDGoC
Study	LangChain 스터디
day of the week	Sun
form	스터디
done	☐

1. 임베딩이란 무엇인가

임베딩(Embedding)은 **자연어 텍스트를 고정 길이의 수치 벡터로 변환하는 과정**이다.

RAG(Retrieval-Augmented Generation) 시스템에서 임베딩은 **문서 분할 이후, 검색 이전**에 위치하며, 문서와 질문을 동일한 벡터 공간(Vector Space) 위에 올려 **의미적 유사도 계산이 가능하도록 만드는 핵심 단계**다.

자연어는 컴퓨터가 직접 비교할 수 없다.

"비슷한 의미"라는 개념을 계산 가능하게 만들기 위해, 임베딩은 텍스트를 **의미가 반영된 숫자의 배열**로 변환한다.

2. RAG 파이프라인에서 임베딩의 역할

RAG 시스템에서 임베딩은 다음 역할을 수행한다.

1. 문서 단위(Chunk)를 벡터 형태로 변환하여 저장한다.
2. 사용자 질문(Query)도 동일한 임베딩 모델로 벡터화한다.
3. 질문 벡터와 문서 벡터 간의 유사도를 계산한다.
4. 가장 유사한 문서를 검색하여 LLM 입력으로 전달한다.

즉, **임베딩의 품질은 곧 검색 품질**이며, 검색 품질은 그대로 **최종 답변의 정확도와 신뢰성**으로 이어진다.

3. 임베딩이 필요한 이유

3.1 의미 이해의 문제

자연어는 단어의 표면 형태보다 **문맥과 의미**가 중요하다.

임베딩 모델은 단어 수준을 넘어 문장·문단의 의미를 벡터 공간의 위치로 표현한다.

예를 들어,

- “안녕하세요? 반갑습니다.”
- “안녕하세요? 만나서 반가워요.”

이 두 문장은 단어가 다르지만 의미는 매우 가깝다.

임베딩은 이 두 문장을 **가까운 벡터 위치**에 배치한다.

3.2 정보 검색의 문제

전통적인 키워드 검색은 단어 일치에 의존한다.

반면, 임베딩 기반 검색은 **의미 유사성**을 기준으로 한다.

그 결과,

- 표현이 달라도
- 언어가 달라도
- 문장이 길거나 짧아도

의미가 유사하다면 검색될 수 있다.

4. 임베딩의 직관적 예시

문서가 다음과 같이 임베딩된다고 가정하자.

- 1번 단락: `[0.1, 0.5, 0.9, ..., 0.1, 0.2]`
- 2번 단락: `[0.7, 0.1, 0.3, ..., 0.5, 0.6]`
- 3번 단락: `[0.9, 0.4, 0.5, ..., 0.4, 0.3]`

사용자 질문도 벡터로 변환된다.

- 질문: “시장조사기관 IDC가 예측한 AI 소프트웨어 시장의 연평균 성장률은?”
- 질문 벡터: `[0.1, 0.5, 0.9, ..., 0.2, 0.4]`

이후 유사도 계산을 통해 가장 가까운 문서를 선택한다.

- 1번: 80% → 선택
- 2번: 30%
- 3번: 25%

5. OpenAIEmbeddings

5.1 개요

OpenAIEmbeddings는 OpenAI에서 제공하는 사전 학습 임베딩 모델을 사용하는 LangChain 인터페이스다.

문서와 쿼리를 동일한 방식으로 벡터화할 수 있으며, RAG에서 가장 널리 사용된다.

5.2 모델 선택

모델	입력 한도	성능(MTEB)	특징
text-embedding-3-small	8191	62.3%	가성비
text-embedding-3-large	8191	64.6%	고성능
text-embedding-ada-002	8191	61.0%	이전 세대

```
from langchain_openaiimport OpenAIEmbeddings

embeddings = OpenAIEmbeddings(model="text-embedding-3-small")
```

5.3 쿼리 임베딩

```
text ="임베딩 테스트를 하기 위한 샘플 문장입니다."
query_result = embeddings.embed_query(text)
```

```
query_result[:5]
```

5.4 문서 임베딩

```
doc_result = embeddings.embed_documents([text])
doc_result[0][:5]
```

5.5 차원 조정

기본 차원은 1536이지만, 필요에 따라 줄일 수 있다.

```
embeddings_1024 = OpenAIEmbeddings(
    model="text-embedding-3-small",
    dimensions=1024
)
len(embeddings_1024.embed_documents([text])[0])
```

차원을 줄이면 저장 공간과 연산 비용은 감소하지만, 의미 표현력도 함께 줄어든다.

6. 임베딩 유사도 계산

임베딩 간 유사도는 보통 **코사인 유사도**를 사용한다.

```
from sklearn.metrics.pairwiseimport cosine_similarity

defsimilarity(a, b):
return cosine_similarity([a], [b])[0][0]
```

실험 결과에서 볼 수 있듯이,

- 같은 의미의 문장은 높은 유사도
- 다른 주제의 문장은 낮은 유사도를 가진다.

7. HuggingFace 기반 임베딩

7.1 Endpoint 기반 임베딩

HuggingFaceEndpointEmbeddings는

외부 HuggingFace Inference Endpoint를 사용해 임베딩을 계산한다.

```
from langchain_huggingface.embeddingsimport HuggingFaceEndpointEmb
eddings

hf_embeddings = HuggingFaceEndpointEmbeddings(
    model="intfloat/multilingual-e5-large-instruct",
    task="feature-extraction",
    huggingfacehub_api_token=os.environ["HUGGINGFACEHUB_API_TOKE
N"],
)
```

다국어 환경에서 특히 강력하다.

7.2 로컬 HuggingFace Embeddings

```
from langchain_huggingface.embeddingsimport HuggingFaceEmbeddings

hf_embeddings = HuggingFaceEmbeddings(
    model_name="intfloat/multilingual-e5-large-instruct",
    model_kwargs={"device":"cuda"},
    encode_kwargs={"normalize_embeddings":True},
)
```

- 로컬 실행
- 빠른 응답
- 데이터 외부 전송 없음

8. BGE-M3와 FlagEmbedding

BGE-M3는 하나의 모델에서 세 가지 검색 방식을 결합한다.

8.1 Dense Vector

의미 기반 검색의 기본 방식.

```
from FlagEmbeddingimport BGEM3FlagModel

bge_embeddings = BGEM3FlagModel("BAAI/bge-m3", use_fp16=True)
bge_embeddings.encode(texts)["dense_vecs"]
```

8.2 Sparse Embedding (Lexical Weight)

정확한 단어 매칭에 유리하다.

```
bge_encoded = bge_embeddings.encode(texts, return_sparse=True)
```

8.3 Multi-Vector (ColBERT)

토큰 단위의 세밀한 매칭을 수행한다.

```
bge_encoded = bge_embeddings.encode(texts, return_colbert_vecs=True)
```

이 조합은 **정확도와 의미 이해를 동시에 확보**할 수 있게 해준다.

9. Upstage Embeddings

Upstage는 국내 환경에 특화된 임베딩 모델을 제공한다.

```
from langchain_upstageimport UpstageEmbeddings

query_embeddings = UpstageEmbeddings(model="solar-embedding-1-large-query")
passage_embeddings = UpstageEmbeddings(model="solar-embedding-1-large-passage")
```

- Query / Passage 전용 모델 분리
- 한국어 성능 우수
- 높은 차원(4096) 제공

10. OllamaEmbeddings

Ollama는 로컬에서 임베딩 모델을 실행할 수 있는 도구다.

```
from langchain_community.embeddingsimport OllamaEmbeddings

ollama_embeddings = OllamaEmbeddings(model="nomic-embed-text")
```

- 인터넷 연결 불필요
- 개인정보 보호에 유리
- 경량 실험 환경에 적합

11. GPT4All Embeddings

GPT4AllEmbeddings는 CPU 기반 로컬 임베딩을 제공한다.

```
from langchain_community.embeddingsimport GPT4AllEmbeddings

gpt4all_embd = GPT4AllEmbeddings()
```

- 완전 로컬 실행
- 차원 384
- 가벼운 테스트용으로 적합