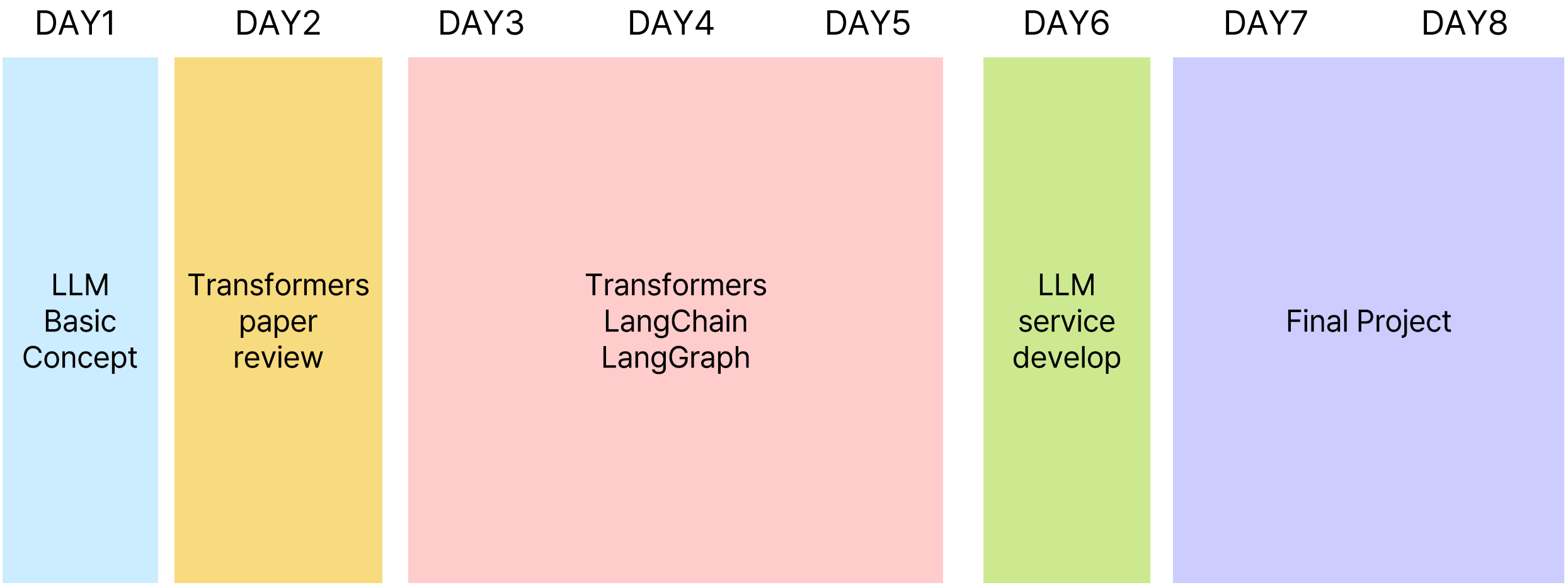
GEN AI 인텐시브 과정

강사 장철원

Section 0

코스소개



Section 0

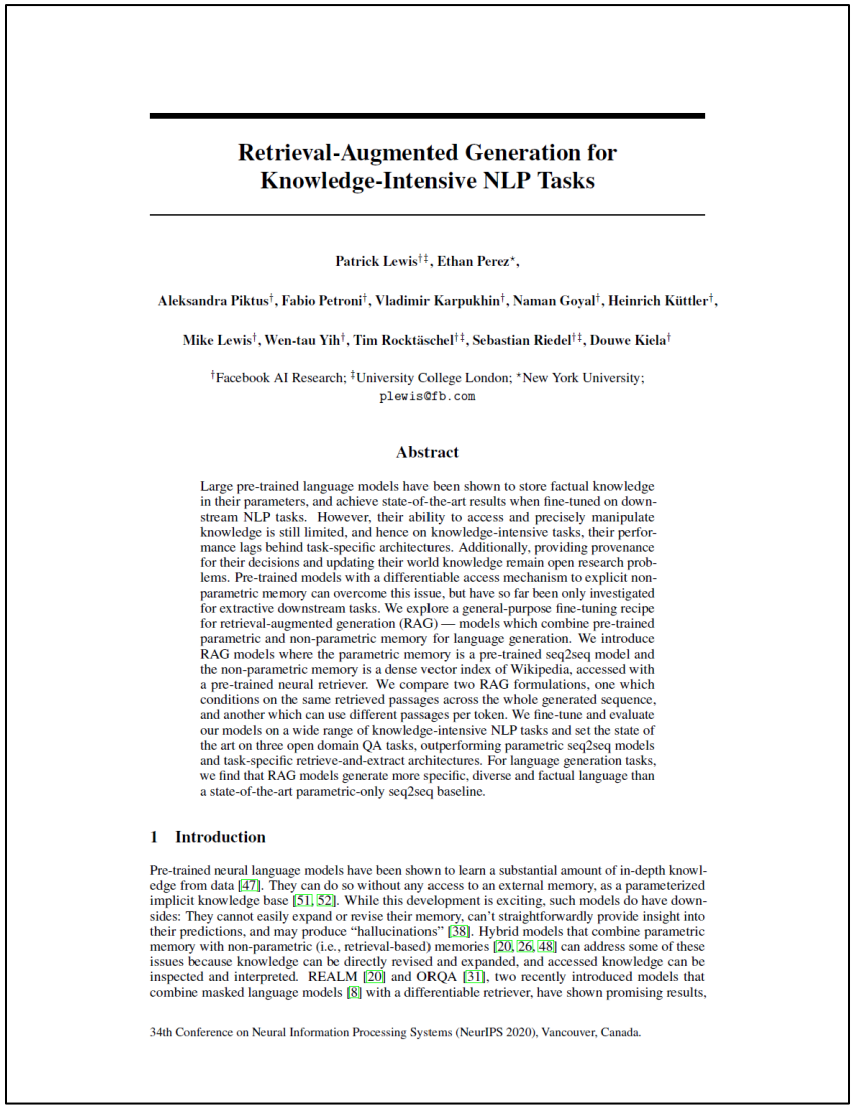
목차

GEN AI 인텐시브 과정

Section 1. RAG 개념

Section 1-1. RAG의 개념

RAG(Retrieval Augmented Generation)



Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis[†‡], Ethan Perez[*],

Aleksandra Piktus[†], Fabio Petroni[†], Vladimir Karpukhin[†], Naman Goyal[†], Heinrich Küttler[†],

Mike Lewis[†], Wen-tau Yih[†], Tim Rocktäschel[†‡], Sebastian Riedel[†‡], Douwe Kiela[†]

[†]Facebook AI Research; [‡]University College London; [*]New York University;
plewis@fb.com

Abstract

Large pre-trained language models have been shown to store factual knowledge in their parameters, and achieve state-of-the-art results when fine-tuned on downstream NLP tasks. However, their ability to access and precisely manipulate knowledge is still limited, and hence on knowledge-intensive tasks, their performance lags behind task-specific architectures. Additionally, providing provenance for their decisions and updating their world knowledge remain open research problems. Pre-trained models with a differentiable access mechanism to explicit non-parametric memory can overcome this issue, but have so far been only investigated for extractive downstream tasks. We explore a general-purpose fine-tuning recipe for retrieval-augmented generation (RAG) — models which combine pre-trained parametric and non-parametric memory for language generation. We introduce RAG models where the parametric memory is a pre-trained seq2seq model and the non-parametric memory is a dense vector index of Wikipedia, accessed with a pre-trained neural retriever. We compare two RAG formulations, one which conditions on the same retrieved passages across the whole generated sequence, and another which can use different passages per token. We fine-tune and evaluate our models on a wide range of knowledge-intensive NLP tasks and set the state of the art on three open domain QA tasks, outperforming parametric seq2seq models and task-specific retrieve-and-extract architectures. For language generation tasks, we find that RAG models generate more specific, diverse and factual language than a state-of-the-art parametric-only seq2seq baseline.

1 Introduction

Pre-trained neural language models have been shown to learn a substantial amount of in-depth knowledge from data [47]. They can do so without any access to an external memory, as a parameterized implicit knowledge base [51, 52]. While this development is exciting, such models do have downsides: They cannot easily expand or revise their memory, can't straightforwardly provide insight into their predictions, and may produce "hallucinations" [38]. Hybrid models that combine parametric memory with non-parametric (i.e., retrieval-based) memories [20, 26, 48] can address some of these issues because knowledge can be directly revised and expanded, and accessed knowledge can be inspected and interpreted. REALM [20] and ORQA [31], two recently introduced models that combine masked language models [8] with a differentiable retriever, have shown promising results,

34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, Canada.

Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

Patrick Lewis et al, 2020

parametric memory VS non parametric memory

LLM이 지식을 어디에 저장하고 활용할지 결정하는 방식

parametric memory

- 모델의 파라미터 안에 내재된 기억. 즉, 모델이 학습을 통해 내부적으로 습득한 지식을 의미
- 예를 들어, GPT와 같은 모델이 학습 데이터로부터 배운 지식을 파라미터(weight)의 형태로 저장하고 활용하는 방식

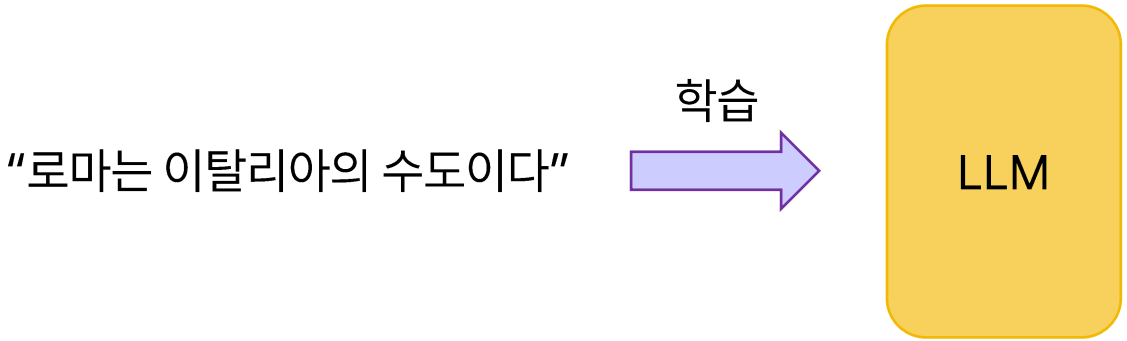
장점

- 출력 속도가 빠르고 항상 사용 가능

단점

- 정보를 업데이트하려면 재학습 해야함
- 특정 시점의 지식에 기반하므로 최신화가 어려움

parametric memory

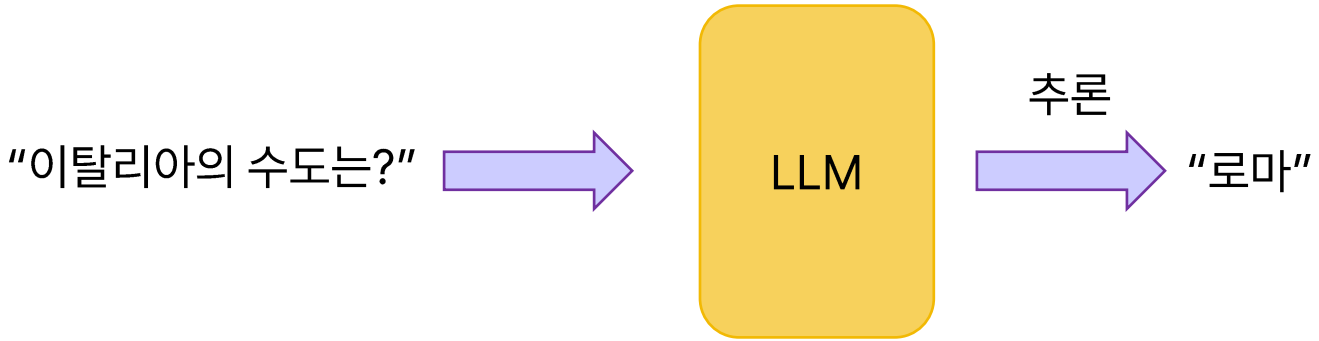


$$w_1 = 0.2, w_2 = 0.3$$

학습을 통해 얻은 정보가 w값에 반영됨

그러나 w값이 구체적으로 어떤 지식을 반영하고 있는지는 해석 불가

parametric memory



$$w_1 = 0.2, w_2 = 0.3$$

추론시에는 모델 내부에 저장된 weight 값을 활용해 응답을 출력

non parametric memory

- 모델 외부에 존재하는 지식에서 검색한 결과를 그대로 답으로 내놓는 방식

장점

- 학습 없이도 응답 가능

단점

- 단순 검색기 수준에 불과함
- parametric 모델과 함께 사용해야 유창한 자연어 응답 가능

RAG = non-parametric memory + parametric memory

- 모델 외부에 존재하는 지식에서 검색해서 활용하는 방식
- 예를 들어, 위키피디아와 같은 웹사이트에서 retrieve가 관련 문서를 찾아오고 이를 기반으로 generator가 응답을 생성하는 방식

장점

- 문서만 바꾸면 최신 지식을 반영할 수 있으므로 업데이트와 유지보수가 쉬움
- 학습 없이도 새로운 정보를 포함할 수 있음

단점

- 외부 지식의 품질에 따라 성능이 좌우됨

RAG

RAG = non parametric memory + parametric memory generator

retriever

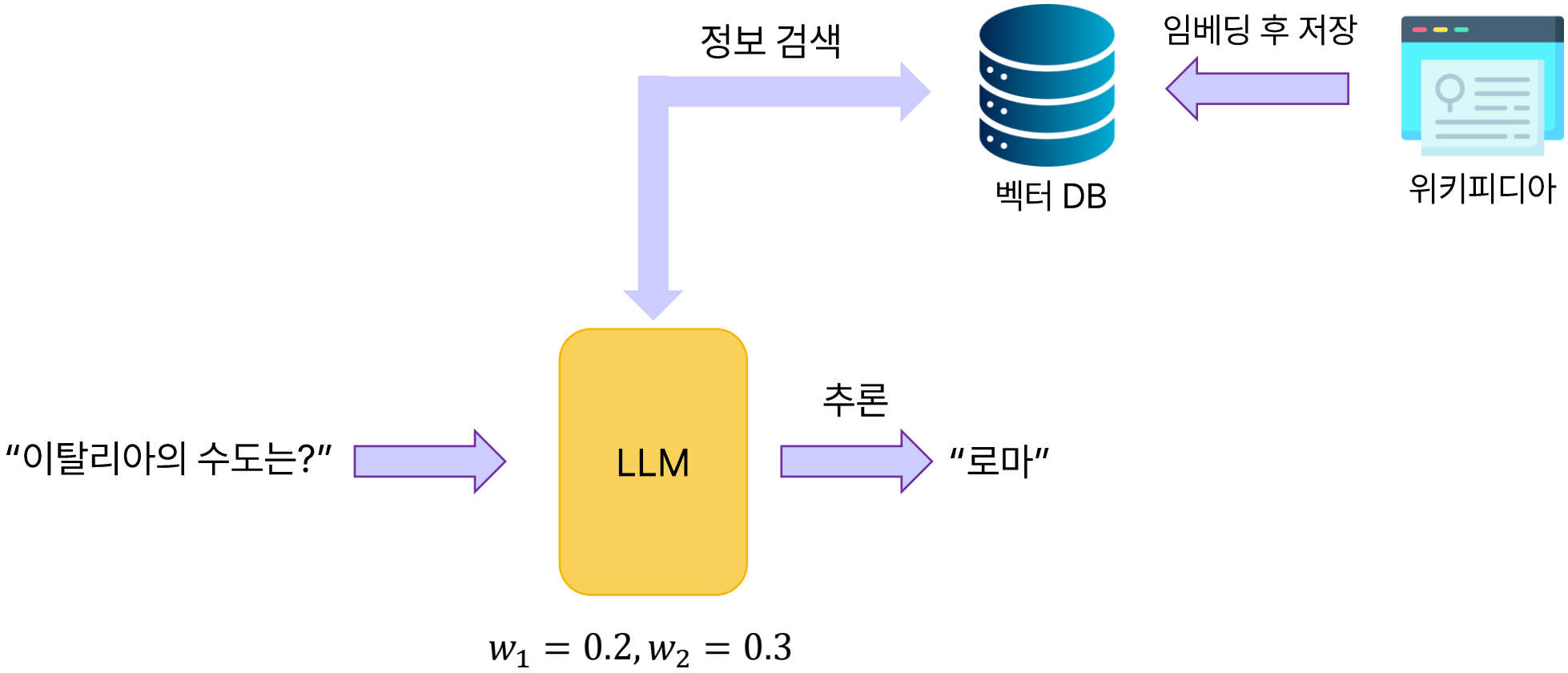
논문에서 사용한 retriever는 사전학습된
Dense Passage Retriever

generator

논문에서 사용한 generator는 사전학습된 BART 모델

* BART: 인코더+디코더 구조를 갖는 트랜스포머 기반 seq2seq 모델

RAG

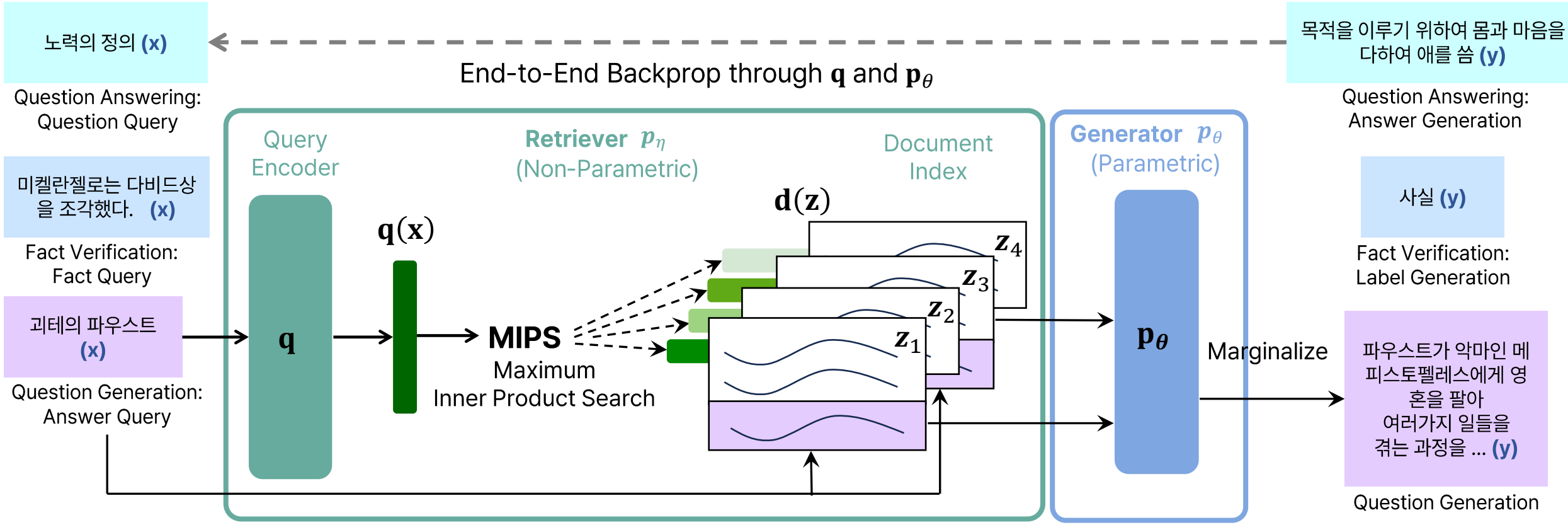


GEN AI 인텐시브 과정

Section 1. RAG 개념

Section 1-1. RAG의 구조

RAG의 구조


노력의 정의 (x)
Question Answering: Question Query
목적을 이루기 위하여 몸과 마음을 다하여 애를 씀 (y)
Question Answering: Answer Generation
End-to-End Backprop through q and p_θ
미켈란젤로는 다비드상을 조각했다. (x)
Fact Verification: Fact Query
사실 (y)
Fact Verification: Label Generation
괴테의 파우스트 (x)
Question Generation: Answer Query
Query Encoder
Retriever p_η (Non-Parametric)
Document Index
Generator p_θ (Parametric)
q
q(x)
MIPS
Maximum Inner Product Search
d(z)
z_4
z_3
z_2
z_1
p_θ
Marginalize
파우스트가 악마인 메피스토펠레스에게 영혼을 팔아 여러가지 일들을 겪는 과정을 ... (y)
Question Generation

Retriever, Generator

Retriever p_η $p_\eta(z|x)$

파라미터가 η일 때, 입력 쿼리 데이터 x가 주어졌을때,
문서 z가 답변에 유용할 확률

Generator p_θ $p_\theta(y_i|x, z, y_{1:i-1})$

파라미터가 θ일 때, 입력 쿼리 데이터 x, retrieved 된 문서 z,
이전 생성 토큰 $y_1, \ldots y_{i-1}$이 주어졌을때,
현재 시점에서의 토큰 y_i가 출력될 확률

RAG를 활용해 출력을 생성하는 두 가지 방법

RAG-Sequence

각각의 출력 토큰을 생성할 때,
모두 동일한 문서를 참고해서 생성하는 방법

RAG-Token

각각의 출력 토큰을 생성할 때,
모두 서로 다른 문서를 참고해서 생성하는 방법

RAG-Sequence Model

특정 문장 y가 생성될 확률

동일한 참고 문서 활용

$$p_{RAG-Sequence}(y|x) \approx \sum_{z \in top-k(p(\cdot|x))} p_\eta(z|x) p_\theta(y|x,z)$$

$$= \sum_{z \in top-k(p(\cdot|x))} p_\eta(z|x) \prod_{i=1}^{N} p_\theta(y_i|x,z,y_{<i})$$

$$\hat{y} = \operatorname*{argmax}_y p_\eta(z|x) p_\theta(y|x,z)$$

RAG-Sequence Model

x 질문: "미켈란젤로의 대표작은?"

z_1 참고 문서1: "미켈란젤로는 피렌체에서 활동했습니다."

z_2 참고 문서2: "그는 다비드상을 남겼습니다."

y_1 "미켈란젤로는 피렌체에서 살았습니다."

y_2 "그의 대표작은 다비드상입니다."

$$p_\eta(z_1|x) = 0.3 \qquad p_\theta(y_1 = \text{"미켈란젤로는 피렌체에서 살았습니다."}|x, z_1) = 0.6$$

$$p_\eta(z_2|x) = 0.7 \qquad p_\theta(y_2 = \text{"그의 대표작은 다비드상입니다."}|x, z_2) = 0.85$$

$$p(y_1|x) = p_\eta(z_1|x)p_\theta(y_1|x, z_1) = 0.3 \times 0.6 = 0.18$$

$$p(y_2|x) = p_\eta(z_2|x)p_\theta(y_2|x, z_2) = 0.7 \times 0.85 = \boxed{0.595}$$

생성될 확률이 높음

$$\hat{y} = \underset{y}{\text{argmax}}\, p_\eta(z|x)p_\theta(y|x, z)$$

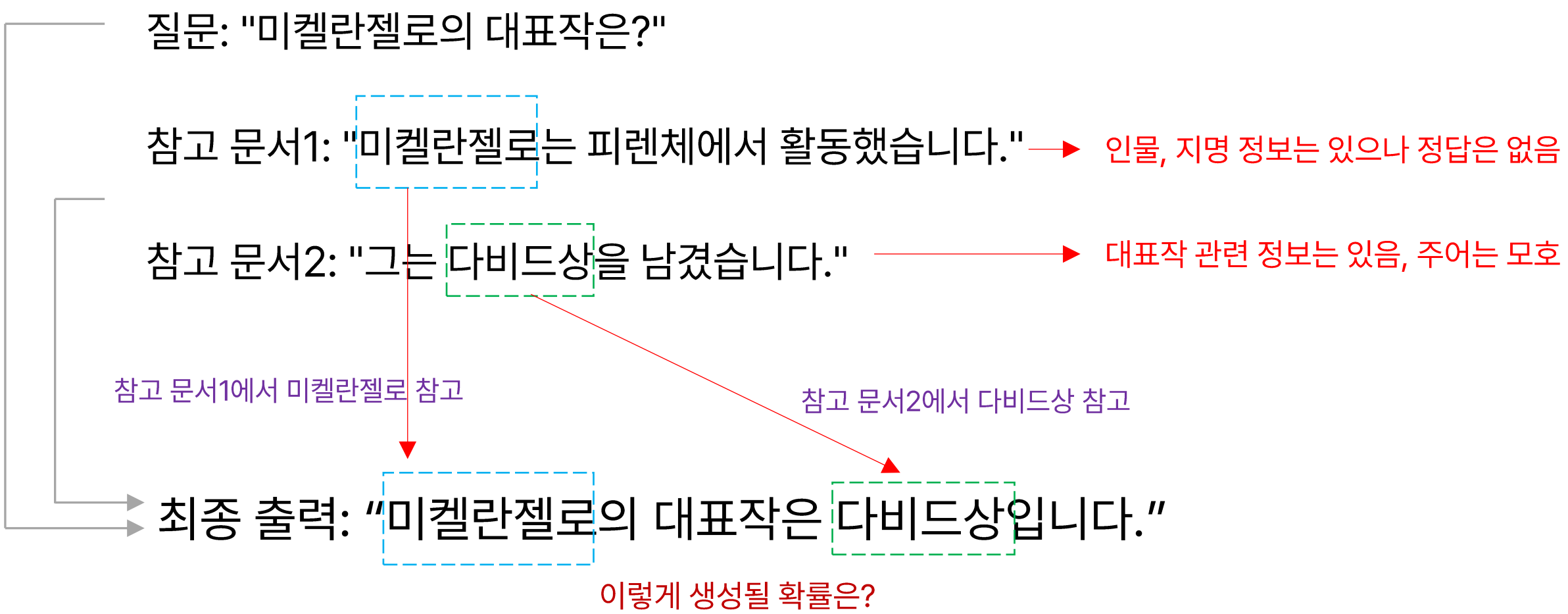
$$\hat{y} = y_2 = \text{"그의 대표작은 다비드상입니다."}$$

RAG-Token Model

특정 문장 y가 생성될 확률

$$p_{RAG-Token}(y|x) \approx \prod_{i=1}^{N} \sum_{k} p_{\eta}(z_k|x) p_{\theta}(y_i|x, z_k, y_{<i})$$

RAG-Token Model



RAG-Token Model

x 질문: "미켈란젤로의 대표작은?"

참고 문서1: "미켈란젤로는 피렌체에서 활동했습니다."

참고 문서2: "그는 다비드상을 남겼습니다."

$$p_\eta(z_1|x) = 0.5$$

$$p_\eta(z_2|x) = 0.5$$

입력 질문에 참고문서1과 참고문서2의
관련성은 동일하다고 가정

최종 출력: "미켈란젤로의 대표작은 다비드상입니다."

출력 토큰

"미켈란젤로의"

"대표작은"

"다비드상입니다."

RAG-Token Model - 첫 번째 토큰 생성

참고 문서	$p_\theta(y_1 = \text{"미켈란젤로의"}\|x, z_i)$	$p_\eta(z_i\|x)$
참고 문서1	$p_\theta(y_1 = \text{"미켈란젤로의"}\|x, z_1) = 0.95$	$p_\eta(z_1\|x) = 0.5$
참고 문서2	$p_\theta(y_1 = \text{"미켈란젤로의"}\|x, z_2) = 0.3$	$p_\eta(z_2\|x) = 0.5$

$$p(y_1 = \text{"미켈란젤로의"}|x)$$

$$= p_\eta(z_1|x)p_\theta(y_1|x, z_1) + p_\eta(z_2|x)p_\theta(y_1|x\ , z_2)$$

$$= 0.5 \cdot 0.95 + 0.5 \cdot 0.3 = 0.625$$

RAG-Token Model - 두 번째 토큰 생성

참고 문서	$p_\theta(y_2 = \text{"대표작은"} \mid x, z_1, y_1)$	$p_\eta(z_i \mid x)$
참고 문서1	$p_\theta(y_2 = \text{"대표작은"} \mid x, z_1, y_1) = 0.6$	$p_\eta(z_1 \mid x) = 0.5$
참고 문서2	$p_\theta(y_2 = \text{"대표작은"} \mid x, z_2, y_1) = 0.8$	$p_\eta(z_2 \mid x) = 0.5$

$$p(y_2 = \text{대표작은} \mid x, y_1)$$

$$= p_\eta(z_1 \mid x) p_\theta(y_2 \mid x, z_1, y_1) + p_\eta(z_2 \mid x) p_\theta(y_2 \mid x, z_1, y_1)$$

$$= 0.5 \cdot 0.6 + 0.5 \cdot 0.8 = 0.7$$

RAG-Token Model - 세 번째 토큰 생성

참고 문서	$p_\theta(y_3 = \text{"다비드상입니다."} \mid x, z_1, y_1, y_2)$	$p_\eta(z_i \mid x)$
참고 문서1	$p_\theta(y_3 = \text{"다비드상입니다."} \mid x, z_1, y_1, y_2) = 0.2$	$p_\eta(z_1 \mid x) = 0.5$
참고 문서2	$p_\theta(y_3 = \text{"다비드상입니다."} \mid x, z_2, y_1, y_2) = 0.9$	$p_\eta(z_2 \mid x) = 0.5$

$$p(y_3 = \text{다비드상입니다.} \mid x, y_1, y_2)$$

$$= p_\eta(z_1 \mid x) p_\theta(y_2 \mid x, z_1, y_1, y_2) + p_\eta(z_2 \mid x) p_\theta(y_2 \mid x, z_2, y_1, y_2)$$

$$= 0.5 \cdot 0.2 + 0.5 \cdot 0.9 = 0.55$$

RAG-Token Model

$$p_{RAG-Token}(y|x) \approx \prod_{i=1}^{3} \sum_{k} p_\eta(z_k|x) p_\theta(y_i|x, z_k, y_{1:i-1})$$

$$\left[\sum_{k} p_\eta(z_k|x) p_\theta(y_1|x, z_k)\right]\left[\sum_{k} p_\eta(z_k|x) p_\theta(y_2|x, z_k, y_1)\right]\left[\sum_{k} p_\eta(z_k|x) p_\theta(y_3|x, z_k, y_1, y_2)\right]$$

$$p(y_1 = \text{"미켈란젤로의"}|x) \times p(y_2 = \text{대표작은} \mid x, y_1) \times p(y_3 = \text{다비드상입니다.} \mid x, y_1, y_2)$$

$$= 0.625 \times 0.7 \times 0.55 = 0.24$$

Retriever: DPR(Dense Passage Retriever)

질문이 주어졌을 때, 해당 질문이 각 문서와 얼마나 연관성이 있는지는 어떻게 알 수 있을까?

$$p_\eta(z|x) \propto \exp\left(\mathbf{d}(z)^{\mathrm{T}}\mathbf{q}(x)\right), \quad \mathbf{d}(z) = BERT_d(z), \quad \mathbf{q}(x) = BERT_q(x)$$

$\mathbf{d}(z)$: Document Encoder를 활용해 문서 z를 인코딩해 dense vector

$\mathbf{q}(x)$: Query Encoder를 활용해 질문 x를 인코딩한 dense vector

GEN AI 인텐시브 과정

Section 2. RAG 개념 실습

Section 2-1. RAG 개념

Section
RAG

라이브러리 불러오기 & 문서 설정

```python
import numpy as np
```

```python
documents = [
    "파이썬은 범용 프로그래밍 언어입니다.",
    "서울은 대한민국의 수도입니다.",
    "RAG는 검색과 생성을 결합한 기술입니다.",
    "GPT는 자연어 생성을 위한 딥러닝 모델입니다.",
    "축구는 세계에서 가장 인기 있는 스포츠입니다."
]

query = "RAG는 무엇인가요?"
```

필요 함수 설정

```python
def tokenizer(text):
    res = []
    text_split = text.replace("?", "").replace(".", "").split()
    for word in text_split:
        res.append(word.lower())
    return res
```

"?" 삭제
"." 삭제
나누기
소문자로 바꿈

```python
example1 = "RAG는 무엇일까요? 알려주세요."
```

```python
example1.replace("?", "")
```

```
'RAG는 무엇일까요 알려주세요.'
```

```python
example1.replace("?", "").replace(".","")
```

```
'RAG는 무엇일까요 알려주세요'
```

```python
example1.replace("?", "").replace(".","").split()
```

```
['RAG는', '무엇일까요', '알려주세요']
```

```python
example1.lower()
```

```
'rag는 무엇일까요? 알려주세요.'
```

Section
RAG

필요 함수 설정

```
def count_vector(tokens, voca):
    res = []
    for word in voca:
        res.append(tokens.count(word))
    return res
```

간단한 예제

```
tokens = ["I", "love", "nlp", "I", "love", "python"]
voca = ["I", "love", "nlp", "deep", "learning"]

vector = count_vector(tokens, voca)
print(vector)
```

```
[2, 2, 1, 0, 0]
```

I 등장 횟수
love 등장 횟수
nlp 등장 횟수
deep 등장 횟수
learning 등장 횟수

```
word = "love"
tokens.count(word)
```

tokens에서 word 등장 횟수

```
2
```

Section
RAG

필요 함수 설정

```
def cosine_similarity(vec1, vec2):
    res = np.dot(vec1, vec2)/(np.linalg.norm(vec1)*np.linalg.norm(vec2))
    return res
```

$$\text{코사인 유사도} = \frac{\langle \mathbf{x}_1, \mathbf{x}_2 \rangle}{\|\mathbf{x}_1\| \|\mathbf{x}_2\|}$$

Section
RAG

토큰 정리

[6]:
```
tokenized_docs = [tokenizer(doc) for doc in documents]
query_tokens = tokenizer(query)
```

[7]:
```
tokenized_docs
```

[7]:
```
[['파이썬은', '범용', '프로그래밍', '언어입니다'],
 ['서울은', '대한민국의', '수도입니다'],
 ['rag는', '검색과', '생성을', '결합한', '기술입니다'],
 ['gpt는', '자연어', '생성을', '위한', '딥러닝', '모델입니다'],
 ['축구는', '세계에서', '가장', '인기', '있는', '스포츠입니다']]
```

[8]:
```
query_tokens
```

[8]:
```
['rag는', '무엇인가요']
```

Section
RAG

토큰 정리

```
voca = sorted(set(query_tokens + sum(tokenized_docs, [])))
```

```
sum(tokenized_docs, [])
```

```
['파이썬은',
 '범용',
 '프로그래밍',
 '언어입니다',
 '서울은',
 '대한민국의',
 '수도입니다',
 'rag는',
 '검색과',
 '생성을',
 '결합한',
 '기술입니다',
 'gpt는',
 '자연어',
 '생성을',
 '위한',
 '딥러닝',
 '모델입니다',
 '축구는',
 '세계에서',
 '가장',
 '인기',
 '있는',
 '스포츠입니다']
```

```
voca
```

```
['gpt는',
 'rag는',
 '가장',
 '검색과',
 '결합한',
 '기술입니다',
 '대한민국의',
 '딥러닝',
 '모델입니다',
 '무엇인가요',
 '범용',
 '생성을',
 '서울은',
 '세계에서',
 '수도입니다',
 '스포츠입니다',
 '언어입니다',
 '위한',
 '인기',
 '있는',
 '자연어',
 '축구는',
 '파이썬은',
 '프로그래밍']
```

쿼리

쿼리

RAG

```
query_vec = count_vector(query_tokens, voca)
similarities = []
for doc_tokens in tokenized_docs:
    doc_vec = count_vector(doc_tokens, voca)
    sim = cosine_similarity(query_vec, doc_vec)
    similarities.append(sim)
```

```
query_vec
```

```
[0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

rag는 무엇인가요

```
tokenized_docs
```

```
[['파이썬은', '범용', '프로그래밍', '언어입니다'],
 ['서울은', '대한민국의', '수도입니다'],
 ['rag는', '검색과', '생성을', '결합한', '기술입니다'],
 ['gpt는', '자연어', '생성을', '위한', '딥러닝', '모델입니다'],
 ['축구는', '세계에서', '가장', '인기', '있는', '스포츠입니다']]
```

```
similarities
```

```
[0.0, 0.0, 0.31622776601683794, 0.0, 0.0]
```

가장 유사함

```
best_doc_index = similarities.index(max(similarities))
retrieved_doc = documents[best_doc_index]
```

```
best_doc_index
```

```
2
```

```
retrieved_doc
```

```
'RAG는 검색과 생성을 결합한 기술입니다.'
```

Section
RAG

답변

```
def generate_answer(query, context):
    res = f"'{context}' 라는 내용을 보면 '{query}'에 대한 답을 유추할 수 있어요."
    return res
```

```
print("질문:", query)
print("관련 문서:", retrieved_doc)
print("생성된 답변:", generate_answer(query, retrieved_doc))
```

```
질문: RAG는 무엇인가요?
관련 문서: RAG는 검색과 생성을 결합한 기술입니다.
생성된 답변: 'RAG는 검색과 생성을 결합한 기술입니다.' 라는 내용을 보면 'RAG는 무엇인가요?'에 대한 답을 유추할 수 있어요.
```

GEN AI 인텐시브 과정

Section 2. RAG 개념 실습

Section 2-1. RAG 기초

라이브러리 불러오기

```
[1]: from transformers import AutoTokenizer, AutoModelForQuestionAnswering
     import torch
```

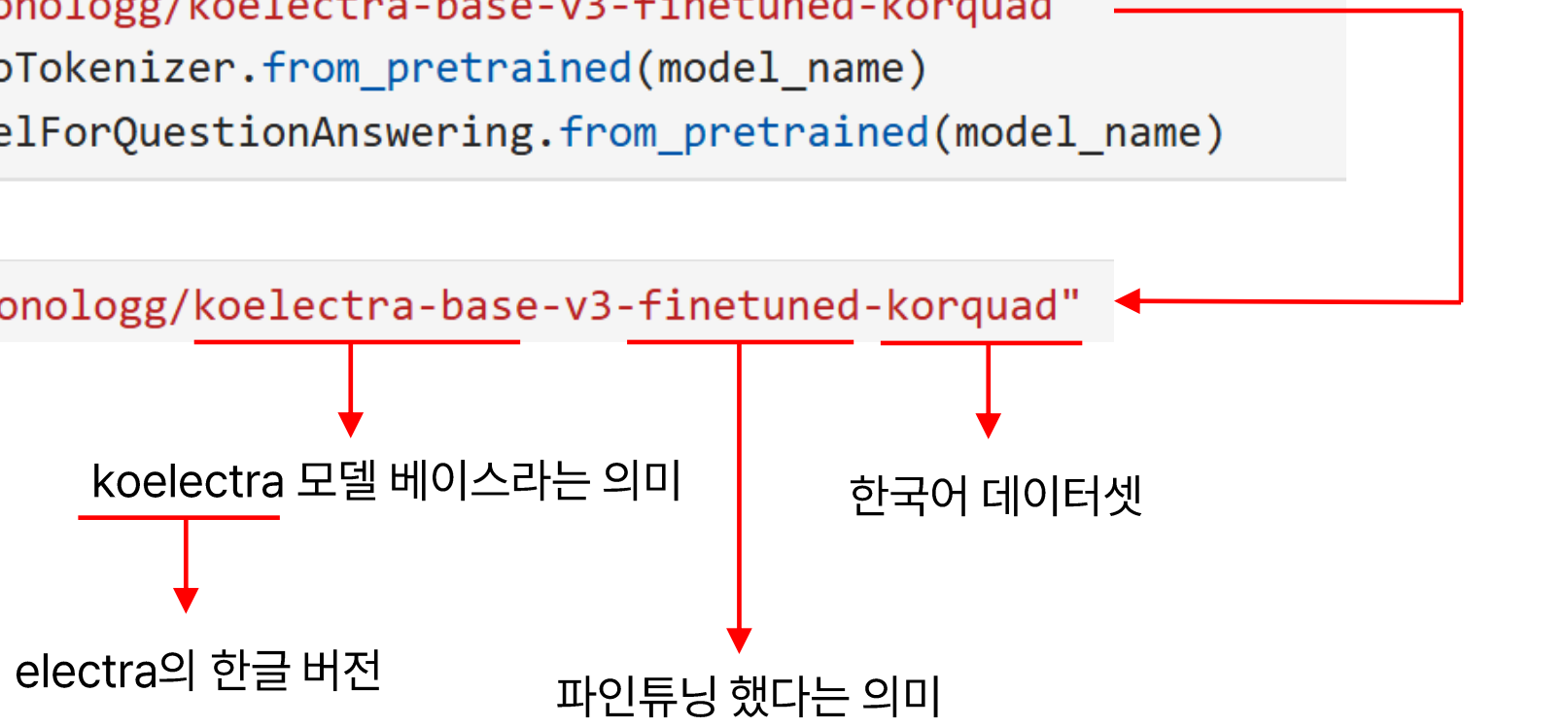
- transformers 라이브러리에서 제공하는 자동 모델 로더 중 하나
- 질문-답변(Question Answering) 태스크를 수행하는 사전 훈련된 모델을 로드하는 역할

AutoModelForQuestionAnswering는 내부적으로 아래 모델 중 하나를 로드(load)

- BertForQuestionAnswering
- RobertaForQuestionAnswering
- DistilBertForQuestionAnswering
- XLNetForQuestionAnswering
- ElectraForQuestionAnswering
- DebertaForQuestionAnswering

모델 이름 설정과 토크나이저 로드

[2]:
```
model_name = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
```



Section
RAG

문서와 질문 설정

[3]:
```
documents = [
    "우리 회사는 인공지능 연구를 하고 있습니다.",
    "우리 제품은 2023년에 출시되었습니다."
]
```

질문에 대한 답을 찾을 문서

[4]:
```
question = "우리 회사는 무슨 연구를 하나요?"
```

질문

문서가 너무 많아지면 DB 필요

Section
RAG

best context 추출

질문에 대한 답을 찾을 문서

질문

```
[5]: def select_best_context(documents, question):
         result = []
         for doc in documents:
             doc_bool = []
             for word in question.split():
                 doc_bool.append(word in doc)
             doc_score = sum(doc_bool)
             result.append(doc_score)
         best_context_idx = result.index(max(result))
         best_context = documents[best_context_idx]
         return best_context
```

documents 내의 문장들 중,
질문과 가장 관련성 높은 문장 추출

질문과 각 문장 간 일치하는 단어가 몇 개인지 확인하고 일치하는 단어가 가장 많은 문장 추출

질문과 각 문장 간 일치하는 단어가 가장 많은 문장 추출

best context 추출

[11]:
```
doc = "우리 회사는 인공지능 연구를 하고 있습니다."
question = "우리 회사는 무슨 연구를 하나요?"
```

→ 문서에 문장이 하나만 존재한다고 가정

→ 질문

[13]:
```
question.split()
```

[13]:
```
['우리', '회사는', '무슨', '연구를', '하나요?']
```

→ split 메서드를 사용하면 해당 문장을 구분자로 나누어 리스트 구조로 만듬
(구분자 디폴트는 공백)
쉽표 기준으로 나누려면 sep=',' 옵션사용

[14]:
```
for word in question.split():
    print(word)
```

→ 질문을 단어별로 쪼개어 출력

```
우리
회사는
무슨
연구를
하나요?
```

best context 추출

```
[15]: doc_bool = []
      for word in question.split():
          doc_bool.append(word in doc)
      print(doc_bool)
```

```
[True, True, False, True, False]
```

문장을 단어별로 나누었을 때 각 단어에 대해 반복문 수행

해당 단어가 문장에 포함되면 True, 포함되지 않으면 False

```
doc = "우리 회사는 인공지능 연구를 하고 있습니다."
question = "우리 회사는 무슨 연구를 하나요?"
```

1 2 3

3개 일치

```
[16]: doc_score = sum(doc_bool)
      print(doc_score)
```

```
3
```

질문을 구성하는 단어들 중 doc 문장에 포함되는 단어 개수
(질문과 doc이 얼마나 일치 하느냐)

best context 추출

변경사항 1 변경사항 2

[17]:
```
doc2 = "우리 회사에서는 인공지능 연구를 하고 있지 않습니다."
question = "우리 회사는 무슨 연구를 하나요?"
```

→ 이번엔 문장을 변경

[18]:
```
doc_bool2 = []
for word in question.split():
    doc_bool2.append(word in doc2)
print(doc_bool)

doc_score2 = sum(doc_bool2)
print(doc_score2)
```

```
[True, True, False, True, False]
2
```

```
doc2 = "우리 회사에서는 인공지능 연구를 하고 있지 않습니다."
question = "우리 회사는 무슨 연구를 하나요?"
```

1 2

이번에는 일치하지 않으므로 False

현 알고리즘의 문제점

```
doc = "우리 회사는 인공지능 연구를 하고 있습니다."
doc2 = "우리 회사에서는 인공지능 연구를 하고 있지 않습니다."

question = "우리 회사는 무슨 연구를 하나요?"
```

이전에는 인공지능 연구를 하고 있다는 의미였는데, 이번 문장은
인공지능 연구를 하지 않는다는 정반대의 의미

따라서 단순하게 단어 일치 개수만으로 관련성을 측정하는 현 알고리즘은 추후 수정 필요

best context 추출

[5]:
```python
def select_best_context(documents, question):
    result = []
    for doc in documents:
        doc_bool = []
        for word in question.split():
            doc_bool.append(word in doc)
        doc_score = sum(doc_bool)
        result.append(doc_score)
    best_context_idx = result.index(max(result))
    best_context = documents[best_context_idx]
    return best_context
```

documents 내의 모든 문장별 일치 개수 모음

documents 내의 모든 문장들에 대해 반복문 수행

가장 큰 일치 개수의 인덱스

일치개수가 가장 많은 문장

Section
RAG

인풋 데이터 생성

[19]:
```
best_context = select_best_context(documents, question)
print(best_context)
```

우리 회사는 인공지능 연구를 하고 있습니다. → documents 내에서 질문과 가장 관련성이 높은 문장

[21]:
```
inputs = tokenizer(question, best_context, return_tensors="pt")
print(inputs)
```

return_tessors="pt" → pytorch
return_tessors="tf" → tensorflow
return_tessors="np" → numpy

```
{'input_ids': tensor([[    2,  6233,  6387,  4034,  7008,  6303,  4110,  6272,  4150,    35,
             3,  6233,  6387,  4034, 11881,  6303,  4110, 14227,  3249,  4576,
          6216,    18,     3]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]),
'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

인풋 데이터 생성

[21]:
```
inputs = tokenizer(question, best_context, return_tensors="pt")
print(inputs)
```

```
{'input_ids': tensor([[    2,  6233,  6387,  4034,  7008,  6303,  4110,  6272,  4150,    35,
             3,  6233,  6387,  4034, 11881,  6303,  4110, 14227,  3249,  4576,
          6216,    18,     3]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'attention_mask': ten
sor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

[24]:
```
inputs.input_ids
```
→ question과 best_context를 합쳐서 하나의 텐서로 표현

[24]:
```
tensor([[    2,  6233,  6387,  4034,  7008,  6303,  4110,  6272,  4150,    35,
             3,  6233,  6387,  4034, 11881,  6303,  4110, 14227,  3249,  4576,
          6216,    18,     3]])
```

[25]:
```
inputs.token_type_ids
```
→ 텐서 내부에서 question과 best_context를 구분

[25]:
```
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

[26]:
```
inputs.attention_mask
```
→ 텐서 내부에서 주목(attention)해야할 부분, 1이면 주목, 0이면 무시

[26]:
```
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

Section
RAG

인풋 데이터 생성

```
[24]: inputs.input_ids
```

→ question과 best_context를 합쳐서 하나의 텐서로 표현

```
[24]: tensor([[    2,  6233,  6387,  4034,  7008,  6303,  4110,  6272,  4150,    35,
                3,  6233,  6387,  4034, 11881,  6303,  4110, 14227,  3249,  4576,
             6216,    18,     3]])
```

2: [CLS] -> 문장의 시작을 알리는 토큰
3: [SEP] -> 문장의 끝을 알리는 토큰

[CLS] "우리 회사는 무슨 연구를 하나요?" [SEP] "우리 회사는 인공지능 연구를 하고 있습니다." [SEP]

2 question 3 context 3

[2, 6233, 6387, 4034, 7008, 6303, 4110, 6272, 4150, 35, 3, 6233, 6387, 4034, 11881, 6303, 4110, 14227, 3249, 4576, 6216, 18, 3]

[CLS] question [SEP] context [SEP]

인풋 데이터 생성

[2, 6233, 6387, 4034, 7008, 6303, 4110, 6272, 4150, 35, 3, 6233, 6387, 4034, 11881, 6303, 4110, 14227, 3249,4576, 6216, 18, 3]

[CLS] question [SEP] context [SEP]

```
[25]: inputs.token_type_ids
```

→ 텐서 내부에서 question과 best_context를 구분

```
[25]: tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

question context

0이면 question, 1이면 context

인풋 데이터 생성

[2, 6233, 6387, 4034, 7008, 6303, 4110, 6272, 4150, 35, 3, 6233, 6387, 4034, 11881, 6303, 4110, 14227, 3249,4576, 6216, 18, 3]

[CLS] question [SEP] context [SEP]

```
[26]: inputs.attention_mask
```

→ 텐서 내부에서 주목(attention)해야할 부분, 1이면 주목, 0이면 무시

```
[26]: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])
```

1이면 주목, 0이면 무시
0인 경우는 주로 패딩된 토큰인데, 이번 예제에서는 패딩된 부분이 없으므로 모든 토큰에 주목

모델 이름 설정과 토크나이저 로드

```
model.eval()
with torch.no_grad():
    outputs = model(**inputs)
```

각 토큰이 정답의 시작위치가 될 가능성을 점수화시킴

```
print(outputs)
```

```
QuestionAnsweringModelOutput(loss=None, start_logits=tensor([[ -4.9601, -11.9210, -11.9915, -12.1289, -11.9584, -12.1299, -12.5216,
         -12.0291, -12.1545, -12.2190,  -4.9601,  -7.5452,  -7.9683,  -7.9850,
          9.0662,  -5.0904,  -7.8677,  -8.6923,  -9.8993,  -9.8247, -10.9598,
         -9.4014,  -4.9601]]), end_logits=tensor([[ -7.9729, -12.8283, -12.7886, -12.8603, -12.9101, -12.5868, -12.5420,
         -12.9244, -12.8554, -12.6807,  -7.9729,  -9.5976,  -9.7077, -10.7093,
          2.0292,   7.7995,  -3.1036,  -7.4832,  -9.2761, -10.3589,  -8.8760,
         -5.3346,  -7.9729]]), hidden_states=None, attentions=None)
```

인덱스 14

인덱스 15

각 토큰이 정답의 끝위치가 될 가능성을 점수화시킴

[29]:
```
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits) + 1
print(start_index)
print(end_index)
```

```
tensor(14)
tensor(16)
```

14번째 토큰이 시작 위치가 될 가능성이 가장 높음

1을 더한 값이 결과로 나와서 실제로는 15번째 토큰이 마지막 위치가 될 가능성이 가장 높음

모델 이름 설정과 토크나이저 로드

[27]:
```
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits) + 1
```

[28]:
```
answer = tokenizer.decode(inputs["input_ids"][0][start_index:end_index], skip_special_tokens=True)

# 결과 출력
print(f"선택된 문서: {best_context}")
print(f"질문: {question}")
print(f"정답: {answer}")
```

```
선택된 문서: 우리 회사는 인공지능 연구를 하고 있습니다.
질문: 우리 회사는 무슨 연구를 하나요?
정답: 인공지능 연구
```

GEN AI 인텐시브 과정

Section 2. RAG 개념 실습

Section 2-2. 단어 임베딩

Section
단어임베딩

임베딩

```
from sentence_transformers import SentenceTransformer
```

임베딩 라이브러리

문장 임베딩 모델 로딩, 텍스트 벡터화

```
model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

```
text = "안녕하세요, 반갑습니다"
embedding = model.encode(text).tolist()
print(embedding)
```

```
[0.15860025584697723, 0.18782837688922882, 0.21552643179893494, 0.0900106206536293, -0.2088
141906023026, -0.12395677715539932, 0.12217465788125992, 0.14111028611660004, -0.4119275516
88340806961, 0.16972672939300537, 0.21761949360370636, -0.004969957284629345, -0.3036306202
34060287476, -0.04174939543008804, 0.024619033560156822, 0.0771401897072792, -0.1875112205
10233688, -0.09735801070928574, -0.06015365570783615, -0.09808235615491867, 0.2365272343158
843, -0.09212759882211685, -0.09780772775411606, -0.0773683413863182, 0.12680231034755707,
5, 0.21559205651283264, 0.0667034462094307, -0.01119158510118723, -0.026993902400135994, 0.
0.051841575652360916, -0.028713105246424675, -0.29151052236557007, 0.16616477072238922, 0.1
25763595104218, -0.08212501555681229, 0.22609353065490723, 0.03251221030950546, 0.074773564
1368408, -0.10468704998493195, -0.07313011586666107, -0.10407399386167526, -0.2208192348486
9196701, 0.03454487398266792, 0.14441482722759247, -0.046122074127197266, -0.09917916357517
194, -0.019158462062478065, 0.1415454000234604, -0.0994078516960144, -0.08040372282266617,
26, 0.06592810899019241, 0.17209461331367493, 0.04821617528796196, -0.07399369031190872, -0
-0.010383589193224907, 0.0095156729221344, -0.07277492433786392, 0.13799931108951569, 0.036
676741674542427, -0.010562494397163391, 0.07445226609706879, -0.028189584612846375, -0.1561
59487426280975, -0.0037361362483352423, -0.12827859818935394, -0.09417415410280228, -0.0986
034142617136240005, 0.07610099017620087, -0.15334777534008026, 0.0052558728493750095, 0.086
-0.07385937124490738, -0.03865443542599678, -0.20465649664402008, 0.010472239926457405, -0.
05333726480603218, 0.16192008554935455, 0.09254675358533859, 0.2873392403125763, 0.05205319
46746969223, -0.049881596118211746, 0.10549893230199814, -0.09687097370624542, -0.17922820
```

Section
단어임베딩

파일 불러와서 임베딩

```
import pandas as pd
from sentence_transformers import SentenceTransformer
```

```
df = pd.read_csv('./test_file.csv', encoding='utf8')
```

```
df
```

	sentence
0	안녕하세요.
1	누구세요?
2	안녕히 가세요.

```
sc_list = df['sentence'].tolist()
```

```
sc_list
```

```
['안녕하세요.', '누구세요?', '안녕히 가세요.']
```

Section
단어임베딩

임베딩

```
model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

```
res = []
for sc in sc_list:
    embedding = model.encode(sc).tolist()
    res.append(embedding)
```

```
print(res)
```

```
08830945193767548, -0.091385617852211, 0.17361406981945038, 0.1261565387248993, 0.110035203
5146491527557, 0.08996763825416565, -0.07002298533916473, -0.015596861951053143, -0.2463191
0706011354923248, -0.15592090785503387, 0.38494348526000977, 0.06817257404327393, 0.1374689
45666491985321, 0.552453339099884, -0.05124004930257797, 0.07389578223228455, -0.2157899737
1138961315, -0.14119285345077515, 0.09877729415893555, -0.031615376472473145, 0.2932605147
041501999, -0.11503718793392181, -0.003756369464099407, -0.28919246792793274, 0.0730240494
329007149, -0.23715347051620483, 0.162056833505630, -0.216256022453308, 0.034054696559906
1468544, -0.04168121516704559, 0.2048763632774353, 0.060456447303295135, -0.375996708869934
8496, -0.29266858100891113, -0.2677391767501831, 0.18005633354187012, 0.2532242238521576, 0
0995, 0.35616832971572876, 0.11574653536081314, -0.2082044929265976, -0.08011460304260254,
2, 0.0999232679605484, 0.26899176836013794, 0.3037497103214264, -0.5305856466293335, -0.170
0.022560227662324905, -0.07628123462200165, -0.09035894274711609, -0.10819779336452484, -0.
6, 0.06667321920394897, 0.01663840375840664, 0.06461058557033539, 0.0290977843105793, -0.08
-0.7072209119796753, 0.2355823516845703, 0.03780223801732063, -0.34913432598114014, -0.2342
4886746406555176, -0.1074855625629425, -0.295912504196167, -0.13553760945796967, -0.1154651
7486369609832764, 0.1164105162024498, 0.1296989917755127, -0.3372374176979065, -0.135106578
61017036438, 0.37813514471054077, -0.1800611913204193, 0.06940987706184387, 0.187129542231
```

GEN AI 인텐시브 과정

Section 2. RAG 개념 실습

Section 2-3. RAG + 단어 임베딩

라이브러리 불러오기 & 모델 설정

```python
import torch
from transformers import AutoTokenizer, AutoModelForQuestionAnswering
from sentence_transformers import SentenceTransformer, util
```

임베딩 라이브러리

문장 임베딩 모델 로딩, 텍스트 벡터화

유틸리티 함수 모음(유사도 계산 등에 사용)

```python
# QA 모델
qa_model_name = "monologg/koelectra-base-v3-finetuned-korquad"
tokenizer = AutoTokenizer.from_pretrained(qa_model_name)
qa_model = AutoModelForQuestionAnswering.from_pretrained(qa_model_name)
```

```python
# 2. 문장 임베딩 모델
embedding_model = SentenceTransformer("sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2")
```

임베딩 모델 설정

Section

RAG + 단어임베딩

문서&질문 임베딩

```python
# 3. 문서
documents = [
    "우리 회사는 인공지능 연구를 하고 있습니다.",
    "우리 제품은 2023년에 출시되었습니다."
]

# 4. 질문
question = "우리 회사는 무슨 연구를 하나요?"
```

```python
doc_embeddings = embedding_model.encode(documents)
query_embedding = embedding_model.encode(question)
```

```
doc_embeddings
```

```
array([[-1.72145262e-01, -3.86104405e-01, -2.13428944e-01,
        -3.98425013e-01, -1.45522803e-01,  4.79151383e-02,
        -4.57954407e-03,  1.97852820e-01,  3.97700217e-01,
         1.204997
        -6.238354
        -4.346799
         6.216445
        -2.027854
        -8.374856
        -1.525112
         2.515148
         1.340826
        -4.337889
        -2.903749
         1.924416
         3.142655
        -2.106764
        -4.199890
         1.164979
        -3.078019
         1.554568
         2.649714
         5.228378
```

```
query_embedding
```

```
array([-0.01725642, -0.06502742, -0.1481036 , -0.12098704, -0.08493589,
       -0.15418048, -0.11554492, -0.0401539 ,  0.12490045,  0.07153846,
        0.02416505,  0.03574743, -0.22066113, -0.13965301,  0.01324866,
       -0.08248807, -0.2540326 , -0.2940846 ,  0.21141812,  0.10532306,
        0.08354471, -0.1495794 ,  0.16214547, -0.10696717, -0.08899061,
        0.10947652,  0.01916375, -0.13665994, -0.09772758, -0.2842331 ,
       -0.18340498, -0.0992927 ,  0.19192839,  0.17882068,  0.06681143,
        0.25931835,  0.19318494, -0.11875024,  0.09661841,  0.08150728,
        0.07408927, -0.17097506,  0.14200553,  0.17854594,  0.10477034,
        0.16608147, -0.14261131, -0.13700263, -0.3908163 , -0.06129955,
       -0.11560778, -0.30922675, -0.00596311, -0.26137137, -0.06395023,
        0.15557528,  0.2765966 ,  0.02946506,  0.05934209, -0.24156997,
        0.17002913, -0.12472679, -0.29219085,  0.186389  , -0.06894048,
       -0.03422314, -0.12953265,  0.27018148, -0.02847639, -0.06592144,
       -0.02042478, -0.24326086, -0.09035207,  0.23170438,  0.02551214,
        0.11156498,  0.16689143,  0.09231529,  0.2267328 , -0.13885036,
        0.25557327,  0.01083626,  0.16092488,  0.26253858, -0.04992906,
        0.26221937, -0.22003224,  0.26517135, -0.01190735, -0.08187884,
        0.07020479, -0.15604515,  0.0287044 , -0.07250194,  0.065428  ,
        0.27776906,  0.00130866, -0.15184802,  0.29076177,  0.5473623 ,
       -0.0841758 ,  0.22449598,  0.02783912, -0.4638888 , -0.27648938,
        0.04527257,  0.19438215,  0.03107038,  0.3578139 ,  0.05427909,
       -0.07761946,  0.0887552 , -0.45469475, -0.17730992,  0.21471746,
       -0.17874053, -0.3703421 ,  0.10368033,  0.17599991,  0.19172701,
```

유사도 측정

```
cosine_similarity = util.cos_sim(query_embedding, doc_embeddings)[0]
print(cosine_similarity)
```

코사인 유사도

```
tensor([0.6351, 0.3094])
```

```
best_idx = cosine_similarity.argmax().item()
print(best_idx)
```

```
0
```

0번째 문장이 가장 관련있음

```
best_context = documents[best_idx]
print(best_context)
```

```
우리 회사는 인공지능 연구를 하고 있습니다.
```

```
# 7. QA 모델 입력 만들기
inputs = tokenizer(question, best_context, return_tensors="pt")
```

Section

RAG+단어임베딩

답변 출력

```
qa_model.eval()
with torch.no_grad():
    outputs = qa_model(**inputs)
```

```
start_index = torch.argmax(outputs.start_logits)
end_index = torch.argmax(outputs.end_logits) + 1
print(start_index)
print(end_index)
```

```
tensor(14)
tensor(16)
```

```
answer = tokenizer.decode(inputs["input_ids"][0][start_index:end_index], skip_special_tokens=True)
```

```
# 9. 결과 출력
print(f"선택된 문서: {best_context}")
print(f"질문: {question}")
print(f"정답: {answer}")
```

```
선택된 문서: 우리 회사는 인공지능 연구를 하고 있습니다.
질문: 우리 회사는 무슨 연구를 하나요?
정답: 인공지능 연구
```

감사합니다.

Q & A