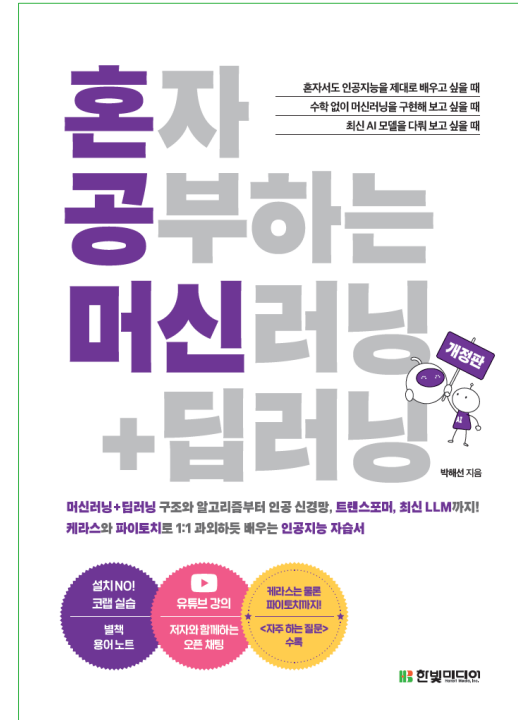


▶ CHAPTER 10 언어 모델을 위한 신경망

# 혼자 공부하는 머신러닝+딥러닝(개정판)



한국공학대학교 게임공학과  
이재영

# 시작하기 전에

## 지은이 / 박해선

기계공학을 전공했으나 졸업 후엔 줄곧 코드를 읽고 쓰는 일을 했다. 머신러닝과 딥러닝에 관한 책을 집필하고 번역하면서 소프트웨어와 과학의 경계를 흥미롭게 탐험하고 있다.

『핸즈온 머신러닝 2판』(한빛미디어, 2020)을 포함해서 여러 권의 머신러닝, 딥러닝 책을 우리말로 옮겼고 『Do it! 딥러닝 입문』(이지스퍼블리싱, 2019)을 집필했다.

- 교재의 모든 코드는 웹 브라우저에서 파이썬 코드를 실행할 수 있는 구글 코랩(Colab)을 사용하여 작성했습니다.
- 사용할 실습 환경은 네트워크에 연결된 컴퓨터와 구글 계정입니다.

# 학습 로드맵



## 머신러닝편

01~06장

딥러닝만 먼저 배우고  
싶다면 01~04장을 읽은 후  
07장으로 건너뛰어도 좋습니다.

## 딥러닝편

07~10장

07장을 읽은 후 08장과 09장은  
순서대로 읽지 않아도 괜찮습니다. 10  
장을 읽기 전에 07장과 09장을  
읽는 것이 좋습니다.

## 난이도



START

01

나의 첫 머신러닝



02

데이터 다루기



03

회귀 알고리즘과 모델 규제



2번 보기

04

다양한 분류 알고리즘



05

트리 알고리즘



06

비지도 학습



07

딥러닝을 시작합니다



08

이미지를 위한 인공 신경망



09

텍스트를 위한 인공 신경망

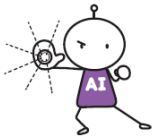


10

언어 모델을 위한 신경망



GOAL



# 이 책의 학습 목표

- **CHAPTER 01: 나의 첫 머신러닝**
  - 인공지능, 머신러닝, 딥러닝의 차이점을 이해합니다.
  - 구글 코랩 사용법을 배웁니다.
  - 첫 번째 머신러닝 프로그램을 만들고 머신러닝의 기본 작동 원리를 이해합니다.
- **CHAPTER 02: 데이터 다루기**
  - 머신러닝 알고리즘에 주입할 데이터를 준비하는 방법을 배웁니다.
  - 데이터 형태가 알고리즘에 미치는 영향을 이해합니다.
- **CHAPTER 03: 회귀 알고리즘과 모델 규제**
  - 지도 학습 알고리즘의 한 종류인 회귀 알고리즘에 대해 배웁니다.
  - 다양한 선형 회귀 알고리즘의 장단점을 이해합니다.
- **CHAPTER 04: 다양한 분류 알고리즘**
  - 로지스틱 회귀, 확률적 경사 하강법과 같은 분류 알고리즘을 배웁니다.
  - 이진 분류와 다중 분류의 차이를 이해하고 클래스별 확률을 예측합니다.
- **CHAPTER 05: 트리 알고리즘**
  - 성능이 좋고 이해하기 쉬운 트리 알고리즘에 대해 배웁니다.
  - 알고리즘의 성능을 최대화하기 위한 하이퍼파라미터 튜닝을 실습합니다.
  - 여러 트리를 합쳐 일반화 성능을 높일 수 있는 앙상블 모델을 배웁니다.

# 이 책의 학습 목표

- **CHAPTER 06: 비지도 학습**
  - 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
  - 대표적인 군집 알고리즘인 k-평균과 DBSCAN을 배웁니다.
  - 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.
- **CHAPTER 07: 딥러닝을 시작합니다**
  - 딥러닝의 핵심 알고리즘인 인공 신경망을 배웁니다.
  - 대표적인 인공 신경망 라이브러리인 텐서플로와 케라스를 소개합니다.
  - 인공 신경망 모델의 훈련을 돕는 도구를 익힙니다.
- **CHAPTER 08: 이미지를 위한 인공 신경망**
  - 이미지 분류 문제에 뛰어난 성능을 발휘하는 합성곱 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 합성곱 신경망을 만들어 패션 MNIST 데이터에서 성능을 평가해 봅니다.
  - 합성곱 층의 필터와 활성화 출력을 시각화하여 합성곱 신경망이 학습한 내용을 고찰해 봅니다.
- **CHAPTER 09: 텍스트를 위한 인공 신경망**
  - 텍스트와 시계열 데이터 같은 순차 데이터에 잘 맞는 순환 신경망의 개념과 구성 요소에 대해 배웁니다.
  - 케라스 API로 기본적인 순환 신경망에서 고급 순환 신경망을 만들어 영화 감상평을 분류하는 작업에 적용해 봅니다.
  - 순환 신경망에서 발생하는 문제점과 이를 극복하기 위한 해결책을 살펴봅니다.
- **CHAPTER 10: 언어 모델을 위한 신경망**
  - 어텐션 메커니즘과 트랜스포머에 대해 배웁니다.
  - 트랜스포머로 상품 설명 요약하기를 학습합니다.
  - 대규모 언어 모델로 텍스트 생성하기를 익힙니다.

- **CHAPTER 10: 언어 모델을 위한 신경망**

SECTION 10-1	어텐션 메커니즘과 트랜스포머
SECTION 10-2	트랜스포머로 상품 설명 요약하기
SECTION 10-3	대규모 언어 모델로 텍스트 생성하기



# CHAPTER 10 언어 모델을 위한 신경망

- 어텐션 메커니즘과 트랜스포머에 대해 배웁니다.
- 트랜스포머로 상품 설명 요약하기를 학습합니다.
- 대규모 언어 모델로 텍스트 생성하기를 익힙니다.

# SECTION 10-1 순차 데이터와 순환 신경망(1)

## ○ 순환 신경망을 사용한 인코더-디코더 네트워크

### - 순환 신경망의 한계

- 시퀀스가 길어질수록 이전에 처리한 데이터를 기억하기 어려움
  - 이 문제를 해결하기 위해 LSTM과 GRU 같은 구조가 개발되었지만, 완벽한 해결책은 아님
- 기계 번역(machine translation) 애플리케이션의 한계
  - 번역할 문장이 길어질수록 기존 RNN 기반 모델은 번역의 품질을 유지하기 어려움
  - 기계 번역에 사용되는 신경망 구조는 전형적으로 시퀀스-투-시퀀스(sequence-to-sequence) 구조  
시퀀스-투-시퀀스 작업은 텍스트를 입력받아 텍스트를 출력
  - 보통 인코더-디코더 구조를 사용하며, 인코더와 디코더에 각각 순환 신경망을 적용

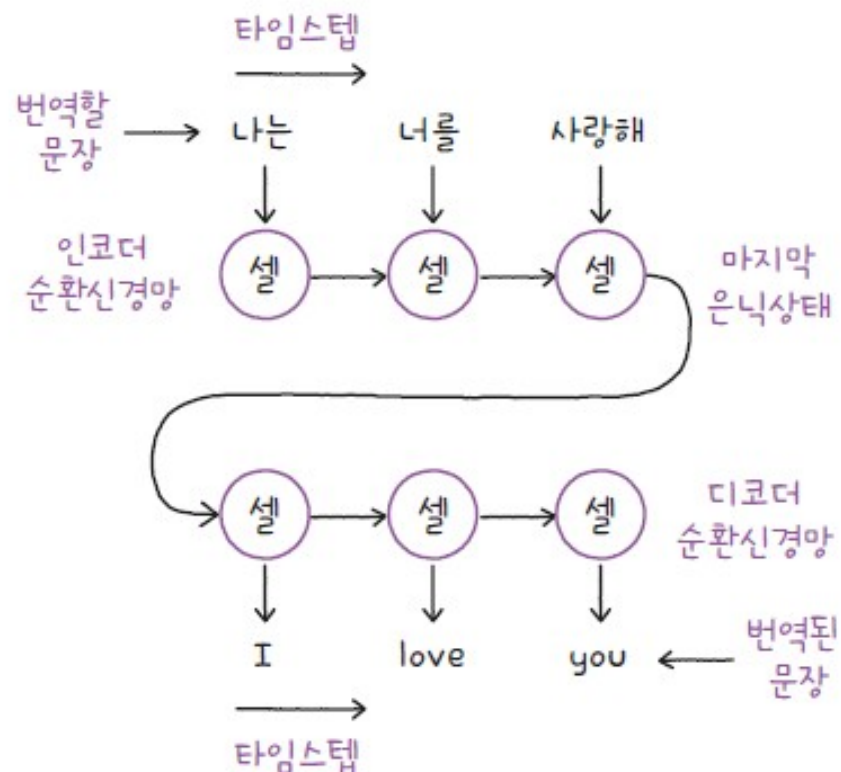


# SECTION 10-1 순차 데이터와 순환 신경망(2)

## ○ 순환 신경망을 사용한 인코더-디코더 네트워크

### - 기계 번역을 수행하는 인코더-디코더 구조

- 1 인코더 신경망은 입력된 문장을 단어(토큰) 단위로 하나씩 처리하면서 전체 정보를 하나의 은닉 상태에 압축
- 2 디코더 신경망이 이 은닉 상태를 받아 마찬가지로 한 단어씩 번역된 문장을 생성



# SECTION 10-1 순차 데이터와 순환 신경망(3)

## ○ 순환 신경망을 사용한 인코더-디코더 네트워크

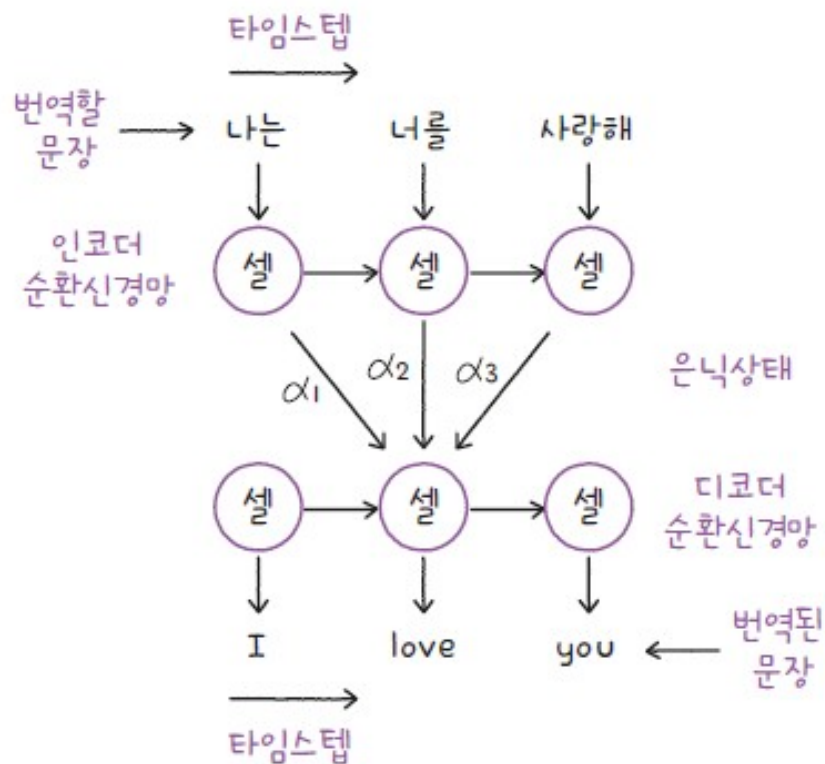
### - 자기회귀 모델(autoregressive model)

- 인코더-디코더 구조에서는 번역할 문장이 길어질수록 초기에 입력된 내용을 기억하기 어려움
- 특히, 디코더 신경망은 인코더의 마지막 은닉 상태만 참고하여 번역을 수행하기 때문에 문제가 더 심해짐
- 또한 인코더와 디코더는 텍스트를 한 토큰씩 처리
  - 입력할 때도 한 토큰씩 받고, 출력할 때도 한 토큰씩 생성해야 하므로 속도가 느림
- 자기회귀 모델
  - 디코더는 “I”라는 토큰을 생성한 후, 인코더의 마지막 은닉 상태와 자신의 은닉 상태를 활용해 “love”를 만듦
  - “I love”를 생성한 후 같은 방식으로 인코더의 마지막 은닉 상태와 자신의 은닉 상태를 활용해 “you”를 출력
  - 이처럼 디코더는 이전에 생성한 토큰을 참고하면서 다음 토큰을 생성

# SECTION 10-1 순차 데이터와 순환 신경망(4)

## ○ 어텐션 메커니즘(attention mechanism)

- 어텐션 메커니즘은 순환 신경망 기반 인코더-디코더 모델의 성능을 크게 향상시킨 기술
  - 기존에는 디코더가 인코더의 마지막 은닉 상태만 참고하여 번역을 수행했지만, 어텐션 메커니즘을 사용하면 인코더의 모든 타임스텝에서 계산된 은닉 상태를 활용



# SECTION 10-1 순차 데이터와 순환 신경망(5)

## ○ 어텐션 가중치

- 디코더가 인코더의 은닉 상태를 활용하는 방식은 가중치를 곱하는 형태로 이루어짐
- 디코더는 인코더의 모든 타임스텝에서 생성된 은닉 상태를 동일하게 참고하는 것이 아니라, 각 은닉 상태마다 가중치를 다르게 적용하여 더 중요한 정보를 강조
  - 이러한 가중치는 다른 모델 파라미터와 마찬가지로 신경망을 훈련하면서 함께 학습
  - (앞 그림) 어텐션 가중치를  $\alpha_1, \alpha_2, \alpha_3$ 으로 나타냄
  - 이 값들은 각각 다른 값을 가지며, 디코더의 타임스텝마다 달라질 수 있음
    - 디코더가 타임스텝에서 인코더의 각기 다른 은닉 상태에 주의를 기울임
- 어텐션 메커니즘은 디코더가 입력 토큰마다 중요도를 다르게 부여하는 방식

# SECTION 10-1 순차 데이터와 순환 신경망(6)

## ○ 어텐션 메커니즘의 장점과 단점

### - 장점

- 어텐션 메커니즘은 긴 텍스트를 처리할 때 정보 손실을 줄이는 데 매우 효과적
- 인코더의 모든 타임스텝에서 생성된 은닉 상태를 참고하여 더 정확한 출력을 생성
- 어텐션 메커니즘을 사용한 획기적인 모델이 등장 - 기계 번역은 물론, 자연어 처리 전 분야에 혁명을 일으킴

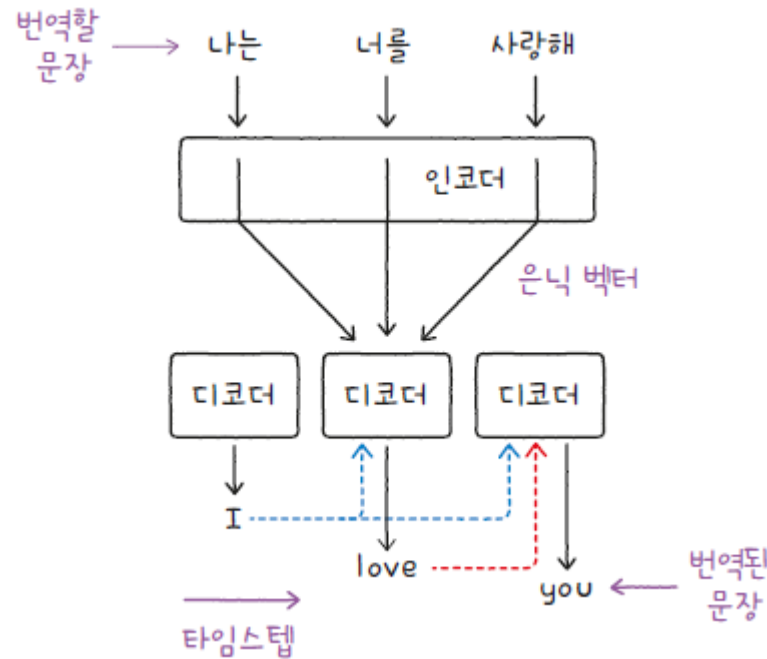
### - 단점

- 어텐션 가중치를 계산하기 위해 인코더의 모든 타임스텝에서 생성된 은닉 상태를 저장 - 연산량이 증가
- 인코더가 처리할 수 있는 타임스텝의 최대 개수를 정해야 하며, 이로 인해 입력 텍스트의 길이가 제한될 수 있음
- 또한, 어텐션을 사용해도 여전히 한 번에 한 토큰씩 처리해야 하는 한계

# SECTION 10-1 순차 데이터와 순환 신경망(7)

## ○ 트랜스포머(Transformer)

- 트랜스포머라는 새로운 신경망 구조 연구는 “Attention Is All You Need”라는 제목의 논문에서 소개
  - 트랜스포머는 논문의 제목처럼 어텐션 메커니즘을 적극적으로 활용
  - 어텐션 메커니즘만 사용하는 것이 아니며 다양한 기술을 조합하여 구성
  - 기존의 인코더-디코더 구조를 유지하면서도 순환 신경망을 완전히 제거  
따라서 입력 텍스트를 한 토큰씩 처리할 필요 없이 한 번에 모두 처리



# SECTION 10-1 순차 데이터와 순환 신경망(8)

## ○ 트랜스포머의 작동 방식

- 트랜스포머는 순환 신경망을 사용하지 않으므로, 더 이상 은닉 상태라는 개념을 사용하지 않음
- 은닉 벡터(hidden vector), 단어 벡터(word vector), 임베딩 벡터(embedding vector)
  - 디코더는 인코더에서 전달받은 은닉 벡터를 활용해 각 타임스텝에서 출력할 토큰을 생성
  - 기존의 인코더-디코더 모델처럼, 디코더는 이전에 생성된 토큰을 참고하면서 새로운 토큰을 생성
  - 순환 신경망 없이도 이전 출력값을 반영할 수 있는 구조
  - 예)
    - 디코더가 두 번째 타임스텝에서 “love”를 출력하려면,
    - 앞서 생성한 텍스트 “I”를 입력으로 받아야 함
    - 세 번째 타임스텝에서는 “I love”를 입력으로 받아 “you”를 출력
- 디코더가 자기회귀 방식으로 작동하는 것은 기존의 순환 신경망을 사용한 인코더-디코더 구조와 동일
  - 하지만 디코더가 이전 출력값을 활용하는 방식이 다름

# SECTION 10-1 순차 데이터와 순환 신경망(9)

## ○ 셀프 어텐션 메커니즘(self-attention mechanism)

### - 기존 어텐션 메커니즘

- 인코더의 은닉 상태와 디코더의 은닉 상태를 비교해 디코더가 특정 타임스텝에서 어떤 입력 토큰에 집중해야 하는지를 학습

### - 트랜스포머

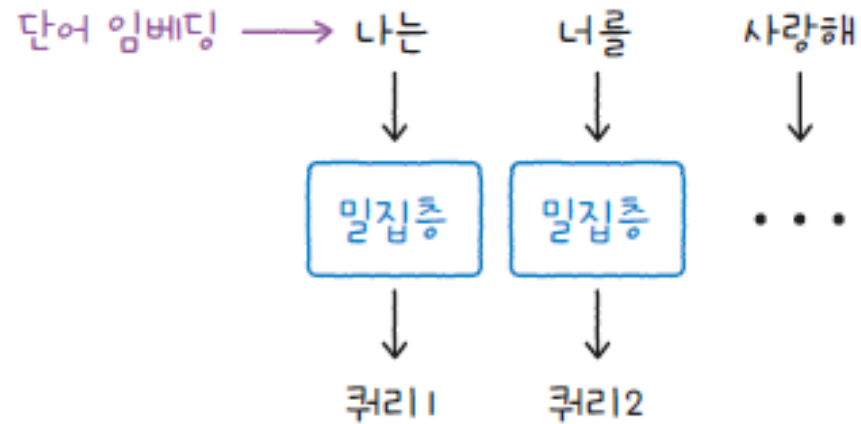
- 인코더에 입력되는 토큰만으로 어텐션 가중치를 학습



# SECTION 10-1 순차 데이터와 순환 신경망(10)

## ○ 셀프 어텐션의 계산 과정

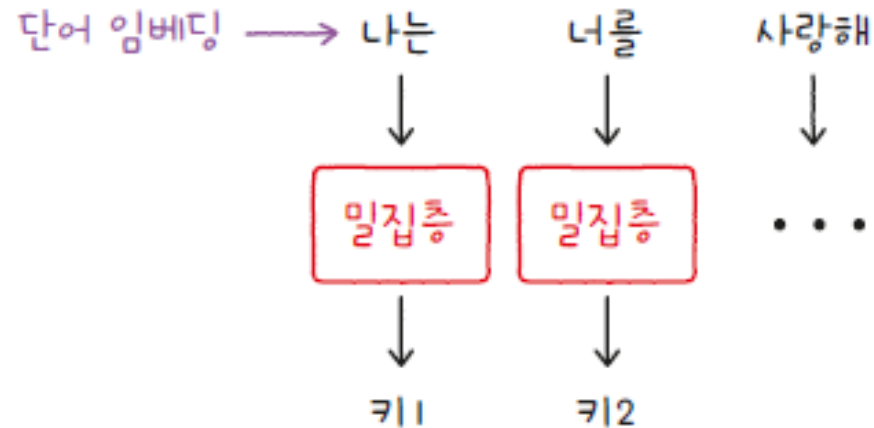
- 입력 텍스트의 각 토큰을 밀집층에 통과시킴(아래 그림에 있는 밀집층은 모두 같은 층)
  - 7장에서 배웠듯이 밀집층은 한 번에 여러 개의 샘플을 처리할 수 있음
  - 여기서는 이해를 돕기 위해 각각의 토큰이 밀집층을 통과하는 것처럼 그려졌지만, 사실 전체 토큰이 한 번에 밀집층을 통과



# SECTION 10-1 순차 데이터와 순환 신경망(11)

## ○ 셀프 어텐션의 계산 과정

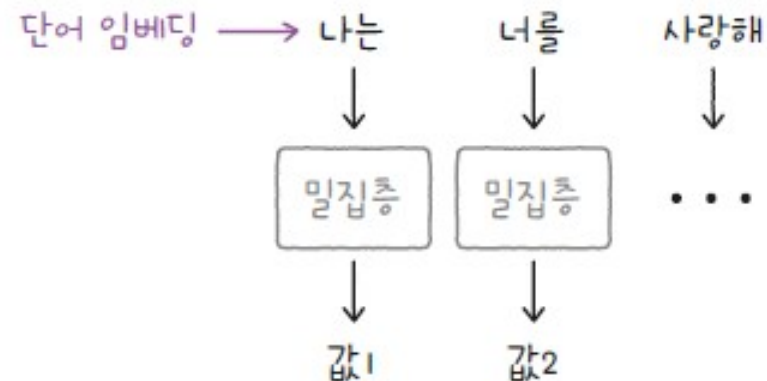
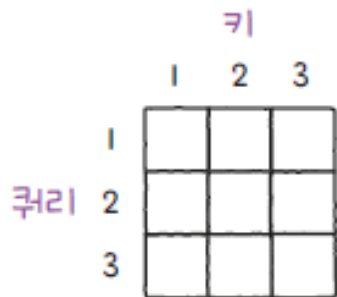
- 쿼리(Query) 벡터
  - 밀집층을 통과한 벡터
- 키(key 벡터)
  - 같은 입력 텍스트를 두 번째 밀집층에 통과시켜 생성
- 쿼리를 생성하는 밀집층과 키를 생성하는 밀집층은 서로 다른 층



## SECTION 10-1 순차 데이터와 순환 신경망(12)

## 어텐션 점수 계산

- 쿼리 벡터와 키 벡터가 생성되면 두 벡터를 서로 곱해서 어텐션 점수(attention score)를 계산
  - 예) 입력된 토큰이 3개라면 쿼리 벡터도 3개, 키 벡터도 3개가 생성  
각 쿼리 벡터와 키 벡터를 곱하면 총 9개의 어텐션 점수가 만들어짐
  - 어텐션 행렬(attention matrix)
- 입력 텍스트를 또 다른 밀집층에 통과시켜 값(value) 벡터를 계산
- 계산된 어텐션 점수를 값 벡터에 곱해서 최종적인 셀프 어텐션 출력을 생성(은닉 벡터)

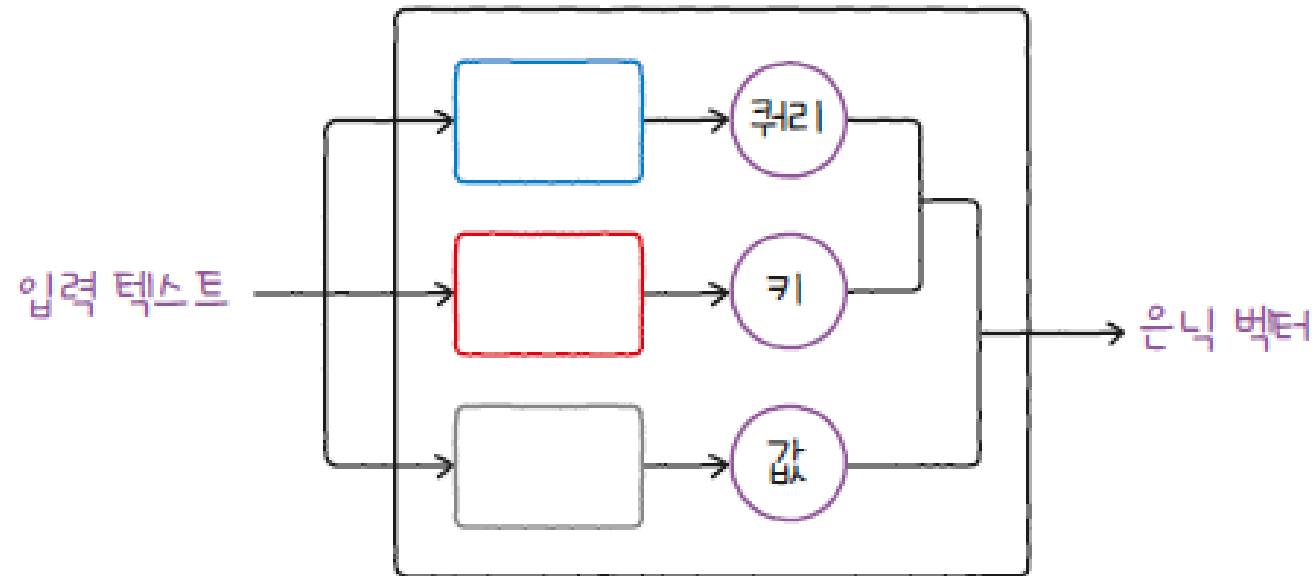


# SECTION 10-1 순차 데이터와 순환 신경망(13)

## ○ 어텐션 점수 계산

### - 셀프 어텐션에서 중요한 점

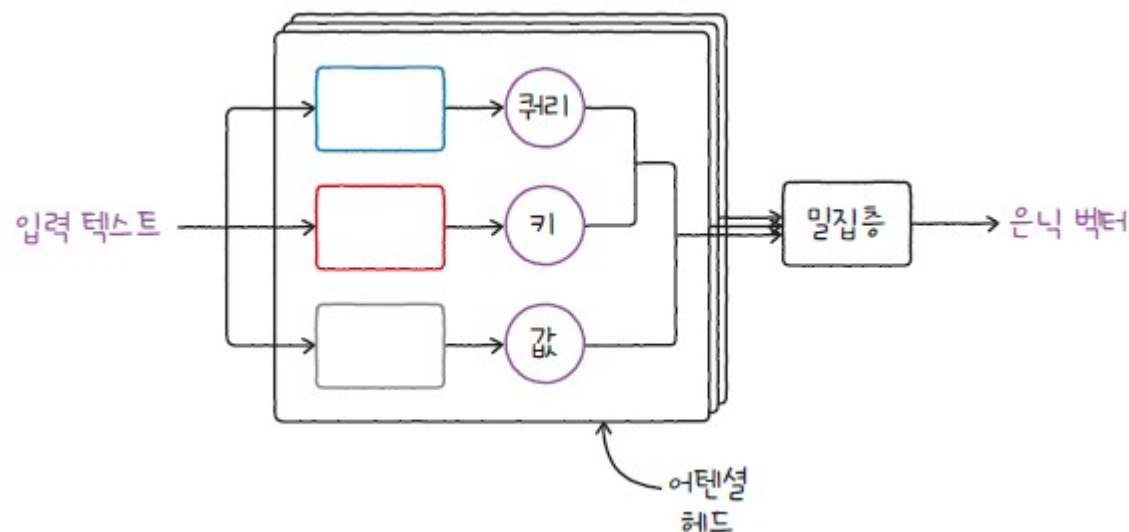
- 각 토큰의 벡터 표현이 주어진 문제를 효과적으로 해결할 수 있도록 학습된다는 것
- 쿼리, 키, 값 벡터를 생성하는 밀집층 세 개의 가중치도 함께 학습됨



# SECTION 10-1 순차 데이터와 순환 신경망(14)

## ○ 멀티 헤드 어텐션

- 어텐션 헤드(attention head) - 셀프 어텐션 연산을 수행하는 하나의
- 멀티 헤드 어텐션(multi-head attention)
  - 단위트랜스포머는 여러 개의 어텐션 헤드를 사용
- 각 어텐션 헤드에서는 쿼리, 키, 값 벡터를 생성하는 밀집층이 서로 다르게 사용
  - 어텐션 헤드들의 출력은 하나로 합쳐진 후, 밀집층을 통과하여 어텐션 층의 최종 출력이 됨
  - 헤드의 개수는 모델마다 다르며, 보통 몇 개에서 많게는 수십 개까지 사용

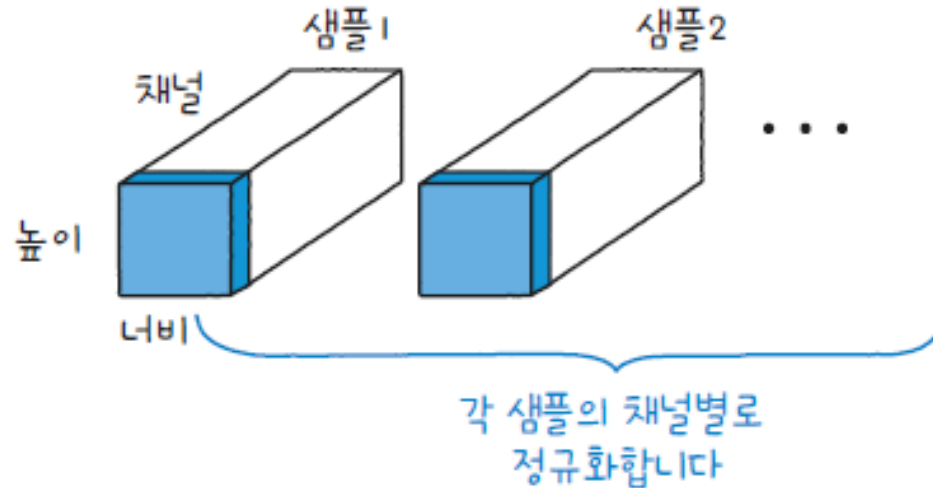


# SECTION 10-1 순차 데이터와 순환 신경망(15)

## ○ 층 정규화

### - 배치 정규화(batch normalization)

- 딥러닝에서는 여러 개의 층을 거치면서 특성의 스케일이 변할 수 있기 때문에, 단순한 입력 정규화만으로는 충분하지 않음
- 배치 정규화는 주로 합성곱 신경망에 널리 활용되며 층과 층 사이에 놓임
- 이전 층의 출력을 배치 단위로 평균과 분산을 계산하여 평균이 0, 분산이 1이 되도록 조정한 후, 다음 층으로 전달
- 모든 샘플에서 특정 채널의 데이터를 모아 평균과 분산을 계산한 후, 정규화를 적용

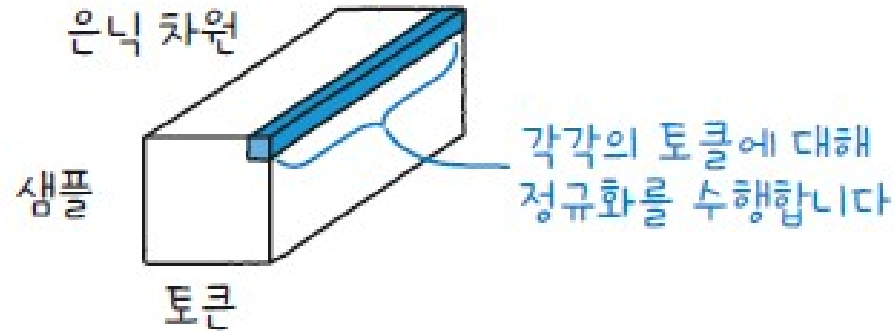


# SECTION 10-1 순차 데이터와 순환 신경망(16)

## ○ 층 정규화

### - 층 정규화(layer normalization)

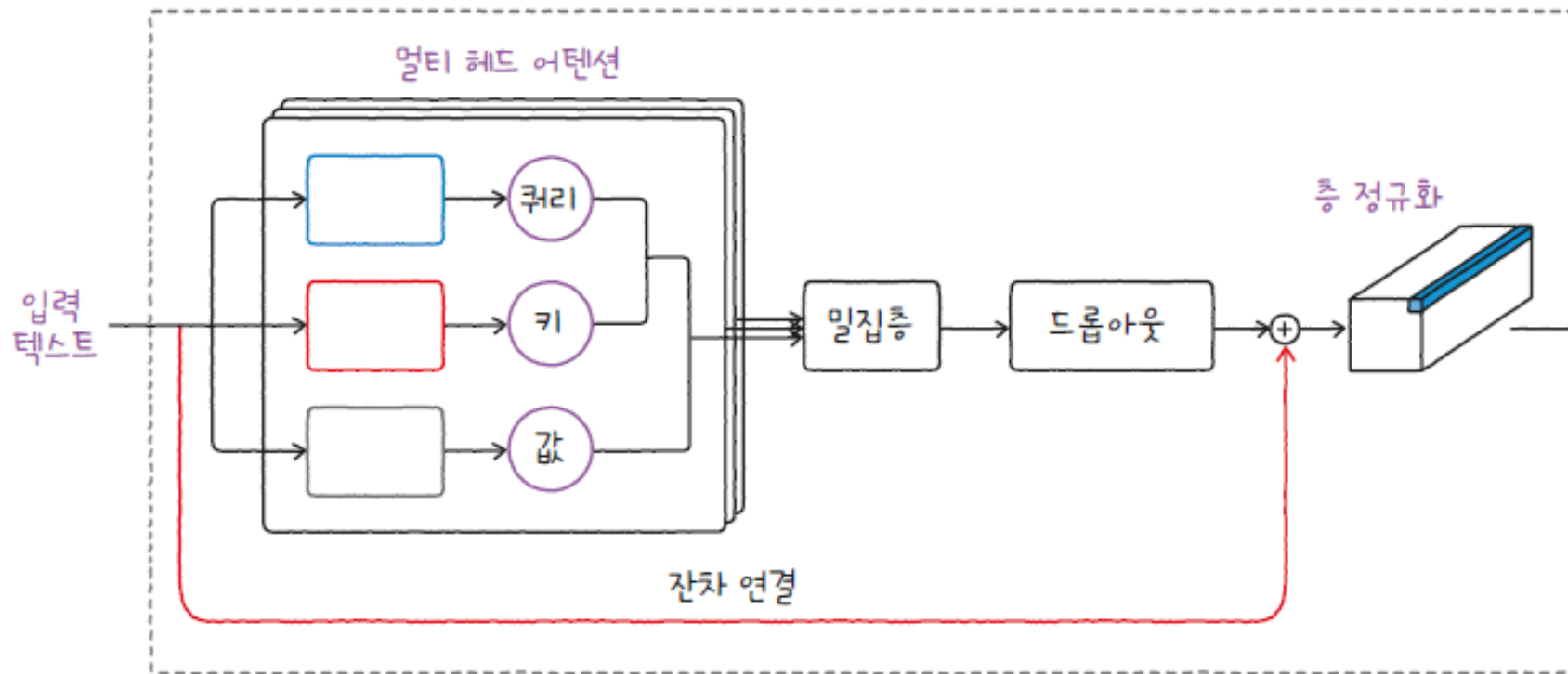
- 배치 정규화를 적용하면 훈련 속도가 빨라지고, 학습 과정이 안정화되며, 이로 인해 모델의 성능이 향상될 수 있어, 많은 신경망에서 널리 사용
  - 하지만 이 방식을 텍스트 데이터는 샘플마다 길이가 다르기 때문에 텍스트 데이터에 적용하기는 어려움
- 층 정규화는 각 샘플의 토큰마다 개별적으로 정규화를 수행
  - 이렇게 하면 샘플마다 길이가 달라도 독립적으로 정규화할 수 있어 샘플의 길이에 영향을 받지 않음



## SECTION 10-1 순차 데이터와 순환 신경망(17)

### ○ 층 정규화

- 트랜스포머에서도 멀티 헤드 어텐션 층 다음에 드롭아웃과 층 정규화가 사용





# SECTION 10-1 순차 데이터와 순환 신경망(18)

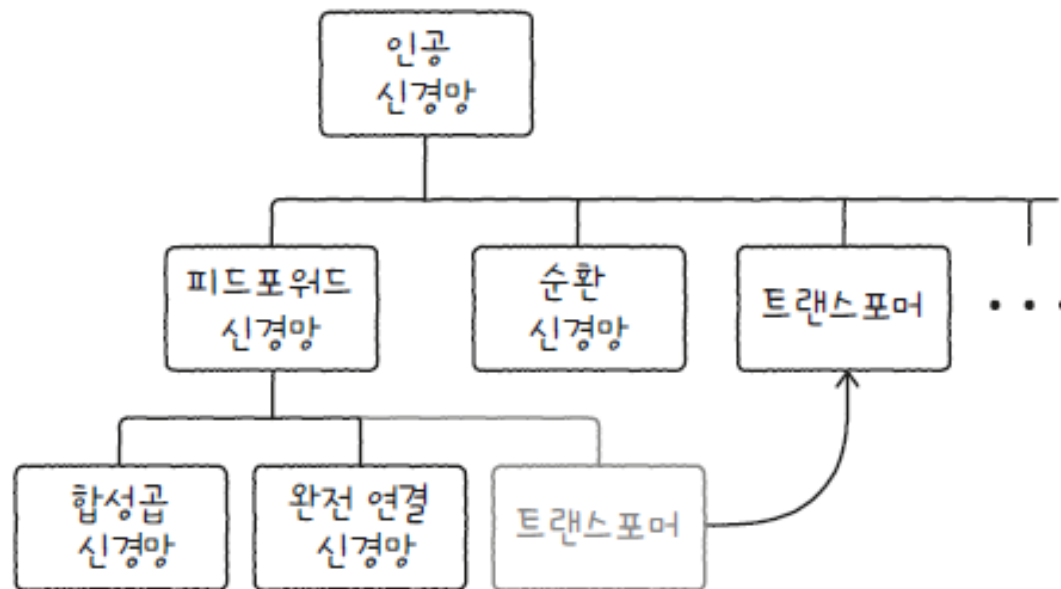
## ○ 층 정규화

- 멀티 헤드 어텐션과 층 정규화 사이에는 잔차 연결(residual connection)이 추가
  - 잔차 연결 또는 스킵 연결(skip connection)은 ResNet이라는 합성곱 신경망에서 처음 도입된 이후, 많은 신경망에서 널리 사용되고 있는 기술
  - 신경망은 층이 많을수록 훈련이 어려워짐
    - 해결하기 위해 잔차 연결이 도입되었으며, 멀티 헤드 어텐션 층을 거친 출력에 입력값을 그대로 더하는 방식
  - 신경망이 훈련될 때는 뒤에서부터 거꾸로 모델의 파라미터 업데이트 신호가 전파
    - 잔차 연결이 추가되면 이 신호가 멀티 헤드 어텐션 층을 거치지 않고 직접 앞쪽 층으로 전달될 수 있음
  - 신경망의 층을 많이 쌓아도 효과적으로 훈련할 수 있음
  - 트랜스포머의 인코더 역시 두 개의 잔차 연결을 사용
    - 두 번째 잔차 연결은 다음에 배울 피드 포워드 네트워크의 앞뒤를 연결하는 역할

# SECTION 10-1 순차 데이터와 순환 신경망(19)

## ○ 피드포워드 네트워크와 인코더 블록

- 피드포워드 신경망에는 합성곱 신경망과 완전 연결 신경망이 포함
  - 기술적으로 보면 트랜스포머 역시 피드포워드 신경망의 한 종류
  - 하지만 기존의 신경망과 구조가 크게 다르고, 많은 파생 모델을 만들어내고 있기 때문에 독립적인 범주로 구분

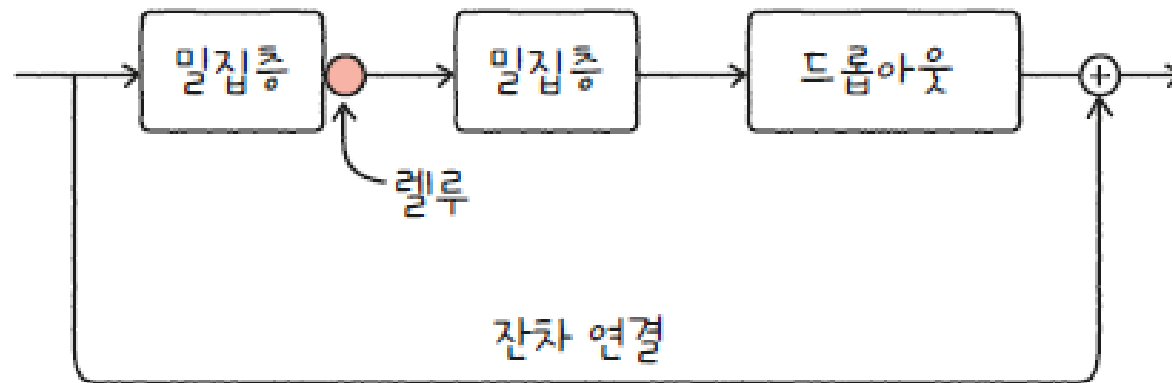


트랜스포머를 추가한 인공 신경망의 종류

## SECTION 10-1 순차 데이터와 순환 신경망(20)

### ○ 피드포워드 네트워크와 인코더 블록

- 피드포워드 네트워크(feedforward network)는 일반적인 피드포워드 신경망을 의미하는 것이 아님
  - 트랜스포머의 인코더에서 멀티 헤드 어텐션과 층 정규화 다음에 나오는 밀집층을 종종 피드포워드 네트워크라고 부름
- 피드포워드 네트워크는 보통 두 개의 밀집층으로 구성
  - 첫 번째 밀집층 - ReLU 활성화 함수를 사용
  - 두 번째 밀집층 - 활성화 함수를 사용하지 않음
  - 그다음 다시 드롭아웃 층이 추가
  - 이 세 개의 층을 또 다른 잔차 연결이 감싸게 됨

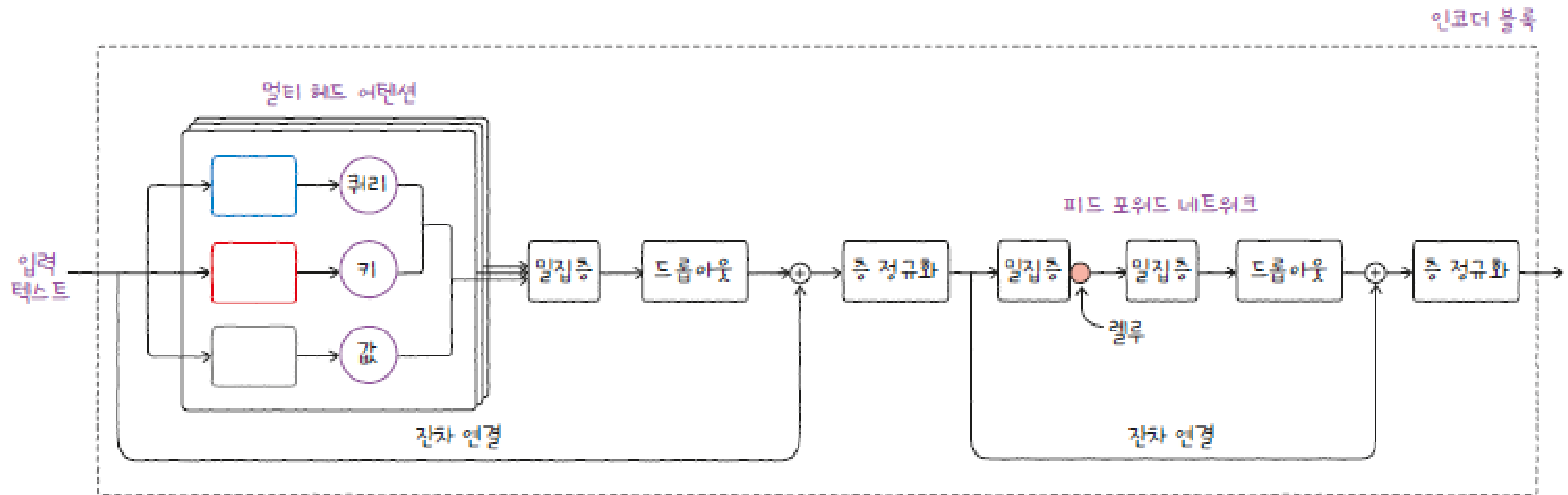


# SECTION 10-1 순차 데이터와 순환 신경망(21)

## ○ 피드포워드 네트워크와 인코더 블록

### - 트랜스포머 인코더 블록

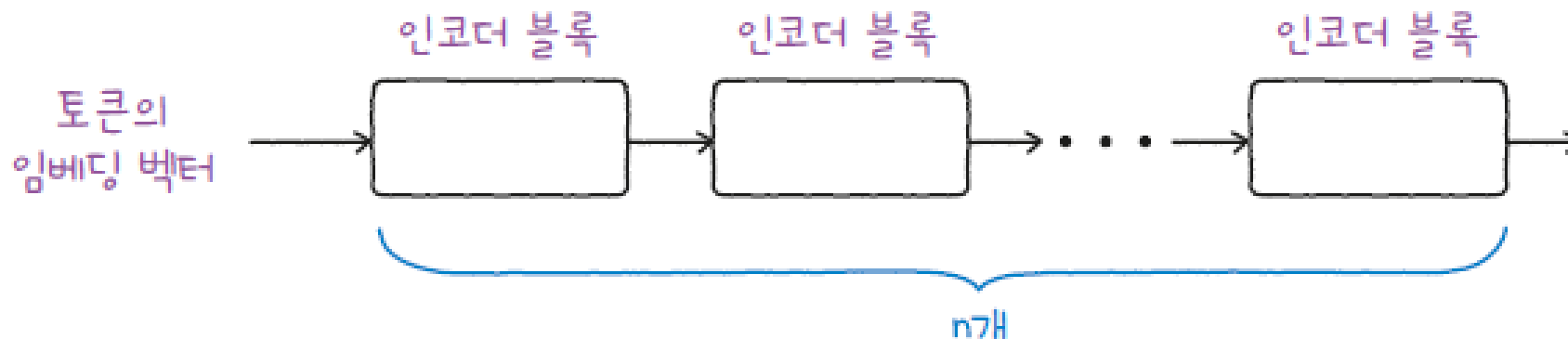
- 멀티 헤드 어텐션과 피드포워드 네트워크를 연결
- 두 부분을 합친 후 마지막에 층 정규화를 다시 배치



## SECTION 10-1 순차 데이터와 순환 신경망(22)

### ○ 피드포워드 네트워크와 인코더 블록

- 인코더 블록이 출력하는 값은 여전히 각 토큰의 은닉 벡터
- 입력 토큰은 단어 임베딩과 같은 벡터 표현으로 변환되어 입력
  - 이 벡터의 차원과 인코더 블록이 출력하는 은닉 벡터의 차원은 동일
  - 이런 특징 덕분에, 동일한 인코더 블록을 여러 개 반복해서 배치할 수 있음
- 적게는 몇 개에서 많게는 수십 개의 인코더 블록을 순차적으로 쌓아 인코더 모델을 구성



## SECTION 10-1 순차 데이터와 순환 신경망(23)

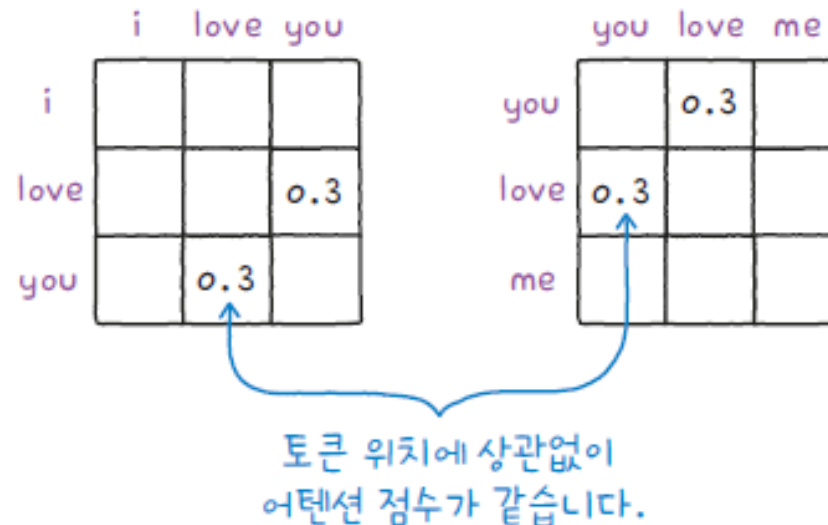
### ○ 토큰 임베딩과 위치 인코딩

- 자연어 처리에서는 모델이 입력된 문자를 이해할 수 있도록 토큰을 숫자로 변환하는 과정이 필요
- 이때 사용되는 대표적인 방법이 단어 임베딩
- 단어 임베딩을 생성하는 방법은 다양하지만, 9장에서 했던 것처럼 임베딩 층을 추가하여 특정 작업에 맞도록 임베딩 벡터를 학습

# SECTION 10-1 순차 데이터와 순환 신경망(24)

## ○ 토큰 임베딩

- 트랜스포머에서도 토큰을 고정된 크기의 실수 벡터로 변환하기 위해 임베딩 층을 사용
  - 트랜스포머는 기존 모델과 다르게 모든 토큰을 동시에 처리하는 방식을 사용하기 때문에 토큰의 위치를 고려하지 않는다는 문제가 발생
  - 앞서 학습한 어텐션 행렬의 계산 과정에서 토큰 간의 관계는 반영되지만 위치는 따로 고려되지 않았음
- 트랜스포머가 문장의 의미를 정확히 이해하려면 위치 정보가 추가적으로 필요
  - 예) “I love you”와 “You love me”라는 두 문장에서 사용된 단어(토큰)는 같지만, 위치가 달라 의미가 완전히 달라짐



# SECTION 10-1 순차 데이터와 순환 신경망(25)

## ○ 위치 임베딩

- 트랜스포머는 위치 인코딩(positional encoding)을 사용
  - 위치 인코딩은 사인(sine) 함수와 코사인(cosine) 함수를 사용해 토큰의 위치에 따라 변하는 벡터를 생성하고, 이를 단어 임베딩에 더하는 방식

1) 예) 임베딩 벡터의 차원이 5이고, 모델에 입력되는 문장의 10번째 토큰이 다음과 같이 표현된다고 가정

10번째  
단어 임베딩

0.1	0.2	0.6	0.3	0.5
-----	-----	-----	-----	-----

2) 벡터의 각 원소 인덱스를 임베딩 벡터의 전체 길이로 나눈 후, 이 값을 10,000의 거듭제곱한 값의 역수로 변환  
그런 다음, 해당 토큰의 순서인 10을 곱함

10번째  
단어 임베딩

0.1	0.2	0.6	0.3	0.5
-----	-----	-----	-----	-----

$$\begin{array}{ccccc} \frac{\text{인덱스}}{\text{길이}} \left( \frac{i}{d} \right) : & \frac{0}{5} & \frac{1}{5} & \frac{2}{5} & \frac{3}{5} & \frac{4}{5} \\ & = 0 & = 0.2 & = 0.4 & = 0.6 & = 0.8 \\ & & & & & - \frac{i}{d} \\ 10 \times 10000 & = 10 & = 1.58 & = 0.25 & = 0.04 & = 0.01 \end{array}$$



# SECTION 10-1 순차 데이터와 순환 신경망(26)

## ○ 위치 임베딩

- 트랜스포머는 위치 인코딩(positional encoding)을 사용

3) 마지막으로 임베딩 벡터에서 짝수 번째 원소의 경우는 사인 함수를 적용하고, 홀수 번째 원소의 경우는 코사인 함수를 적용 - 이렇게 구한 값을 원본 임베딩 벡터에 더함

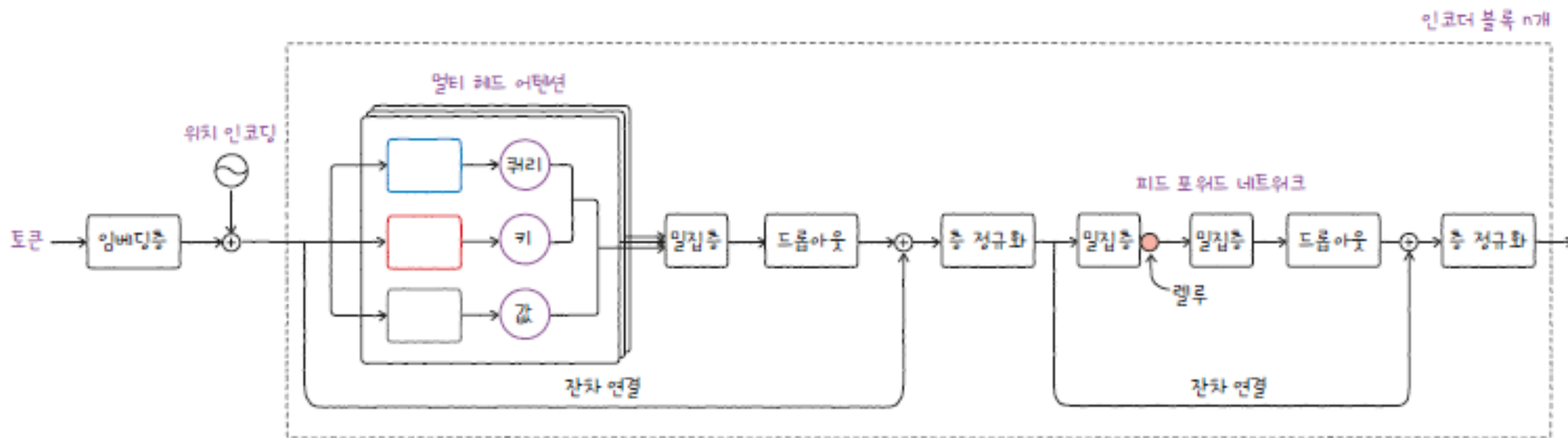
10번째 단어 임베딩	0.1	0.2	0.6	0.3	0.5
$\frac{\text{인덱스}}{\text{길이}} \left( \frac{i}{d} \right) :$	$\frac{0}{5}$	$\frac{1}{5}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{4}{5}$
	=0	=0.2	=0.4	=0.6	=0.8
$10 \times 10000^{-\frac{i}{d}}$	=10	=1.58	=0.25	=0.04	=0.01
sin	=-0.54		=0.24		=0.01
cos		=-0.01		=1.0	
	-0.44	0.19	0.84	1.04	0.81

두 값을 더하여  
최종 단어 임베딩을  
만듭니다.

# SECTION 10-1 순차 데이터와 순환 신경망(27)

## ○ 위치 임베딩

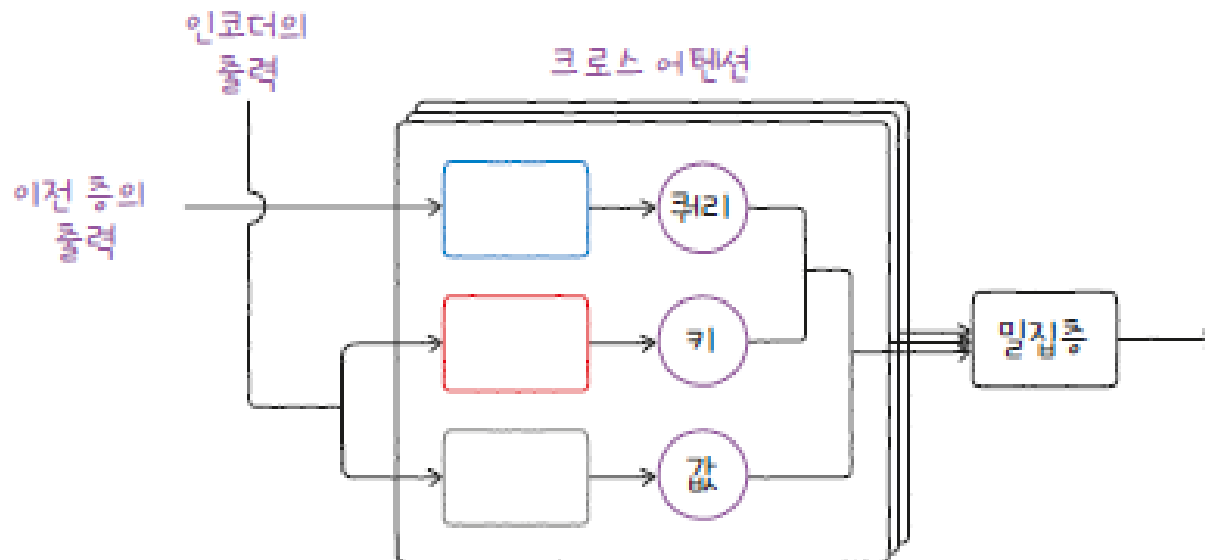
- 절대 위치 인코딩
  - 위치 인코딩은 토큰의 위치와 임베딩 벡터의 차원에 따라 일정한 값으로 계산
- 상대 위치 인코딩(또는 상대 위치 임베딩) / 10-3절에서 학습
- 위치 임베딩의 구성



# SECTION 10-1 순차 데이터와 순환 신경망(28)

## ○ 디코더 블록

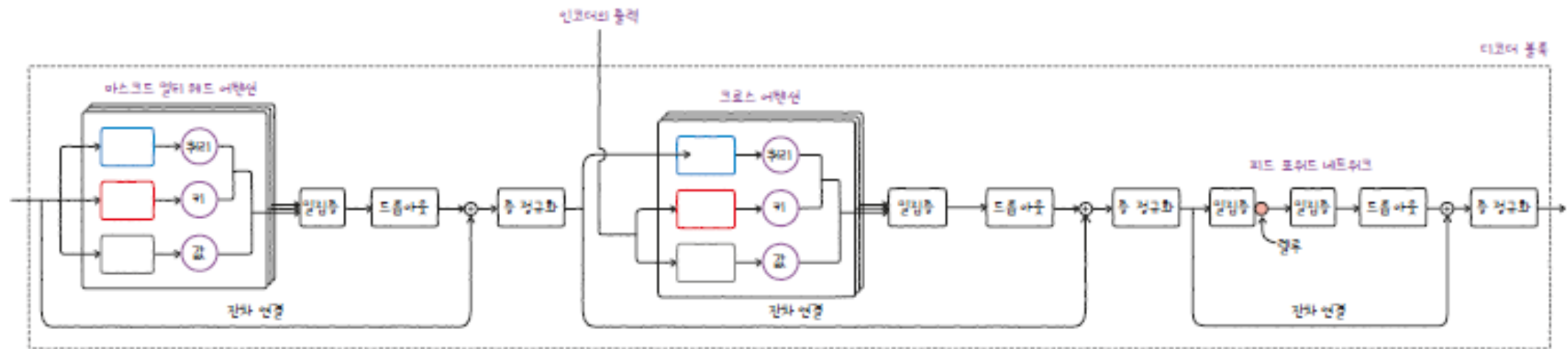
- 트랜스포머 디코더는 몇 가지 차이점을 제외하면 인코더와 매우 비슷함
  - 그중 하나가 인코더가 출력한 임베딩 벡터를 입력으로 받는 멀티 헤드 어텐션 층
  - 이 층은 디코더에서 받은 벡터를 쿼리로 사용하고, 인코더의 출력을 키와 값으로 사용
    - 크로스 어텐션(cross attention)



## SECTION 10-1 순차 데이터와 순환 신경망(29)

### ○ 디코더 블록

- 디코더 블록에서는 크로스 어텐션 층이 인코더의 멀티 헤드 어텐션 층과 피드포워드 네트워크 사이에 배치
- 크로스 어텐션 층의 전후에도 잔차 연결이 있음

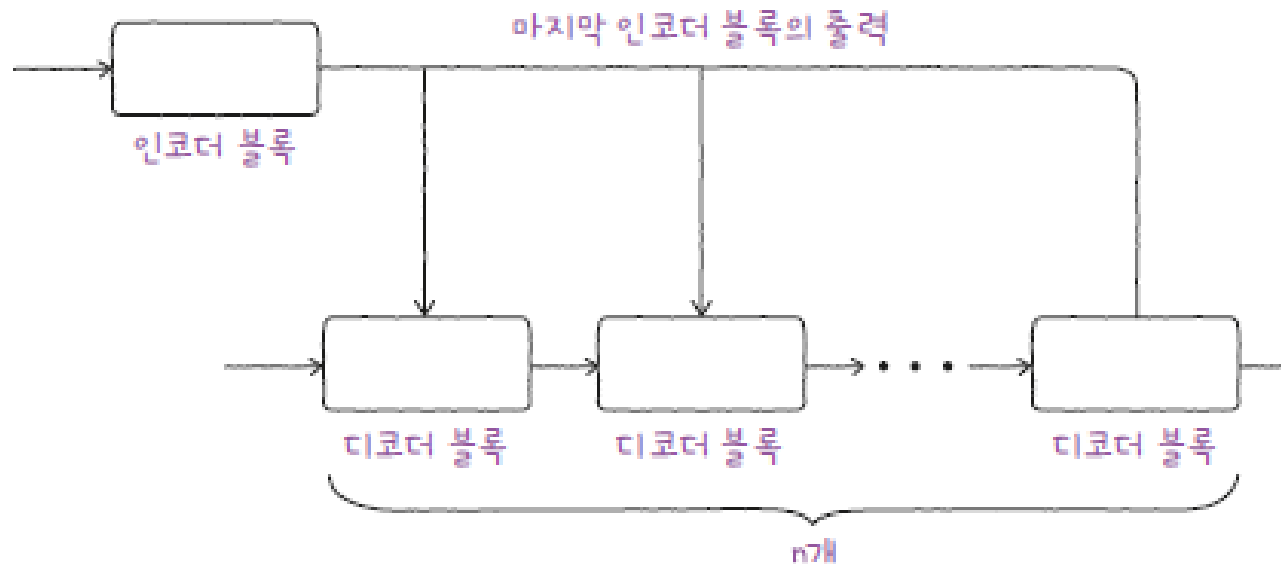


크로스 어텐션을 추가한 디코더 블록

## SECTION 10-1 순차 데이터와 순환 신경망(30)

### ○ 디코더 블록

- 디코더 블록 역시 하나만 존재하는 것이 아니라 여러 개가 반복적으로 쌓여 전체 디코더 모델을 구성
  - 따라서 인코더 블록의 출력은 첫 번째 디코더 블록뿐만 아니라 반복되는 모든 디코더 블록에 전달



# SECTION 10-1 순차 데이터와 순환 신경망(31)

## ○ 디코더 블록

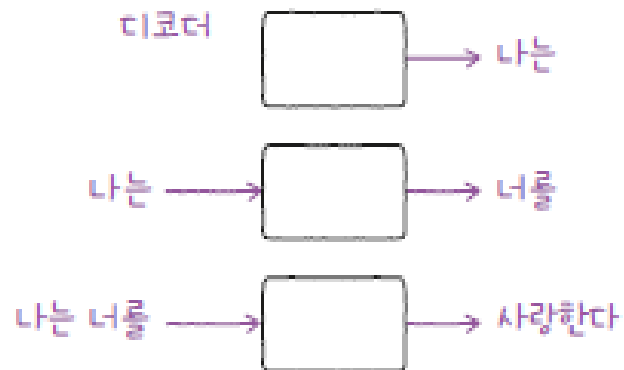
- 디코더 블록이 출력하는 값도 인코더 블록과 마찬가지로 동일한 크기의 은닉 벡터
  - 따라서 여러 개의 디코더 블록을 반복해서 쌓을 수 있음
  - 그다음 해결하고자 하는 작업에 맞는 층을 마지막 디코더 블록 다음에 놓음
  - 예) 텍스트 분류 작업에서는 마지막 디코더 블록 뒤에 밀집층을 추가해 여러 클래스 중 하나를 예측하도록 설계

# SECTION 10-1 순차 데이터와 순환 신경망(32)

## ○ 디코더 블록

### - 디코더 블록과 인코더 블록의 또 다른 차이점

- 맨 왼쪽의 멀티 헤드 어텐션에서 디코더 입력을 처리
- 예) 한-영 번역 모델
  - 인코더의 입력이 “I love you”와 같은 영어 문장이라면, 이 문장 전체가 인코더에 전달
    - 첫 번째 타임스텝에서 디코더는 “나는”을 출력
  - 인코더는 각 토큰을 분석하여 각각의 의미를 포함한 은닉 벡터를 만들고, 이를 디코더 블록에게 전달
    - 두 번째 타임스텝에서는 “나는”이 입력되어 “너를”을 출력
  - 하지만 디코더는 자기회귀 모델의 방식을 따라 한 번에 하나의 토큰만 생성
    - 번째 타임스텝에서는 “나는 너를”이 입력되어 “사랑한다”가 출력



# SECTION 10-1 순차 데이터와 순환 신경망(33)

## ○ 디코더 블록

- 모델을 훈련할 때의 상황은 실제 번역을 수행할 때와 다름
  - “I love you”의 올바른 번역(타깃값)인 “나는 너를 사랑한다”를 디코더에게 한 번에 전달하고, 각 토큰에서 다음 토큰을 예측하도록 모델을 훈련
  - 디코더는 “나는”에서 “너를”을 예측하고, “나는 너를”을 사용해 “사랑한다”를 예측하도록 훈련
- 이 훈련 과정은 모델의 출력을 정답(타깃)과 비교하여 오차를 줄이는 기존의 방식과 동일



# SECTION 10-1 순차 데이터와 순환 신경망(34)

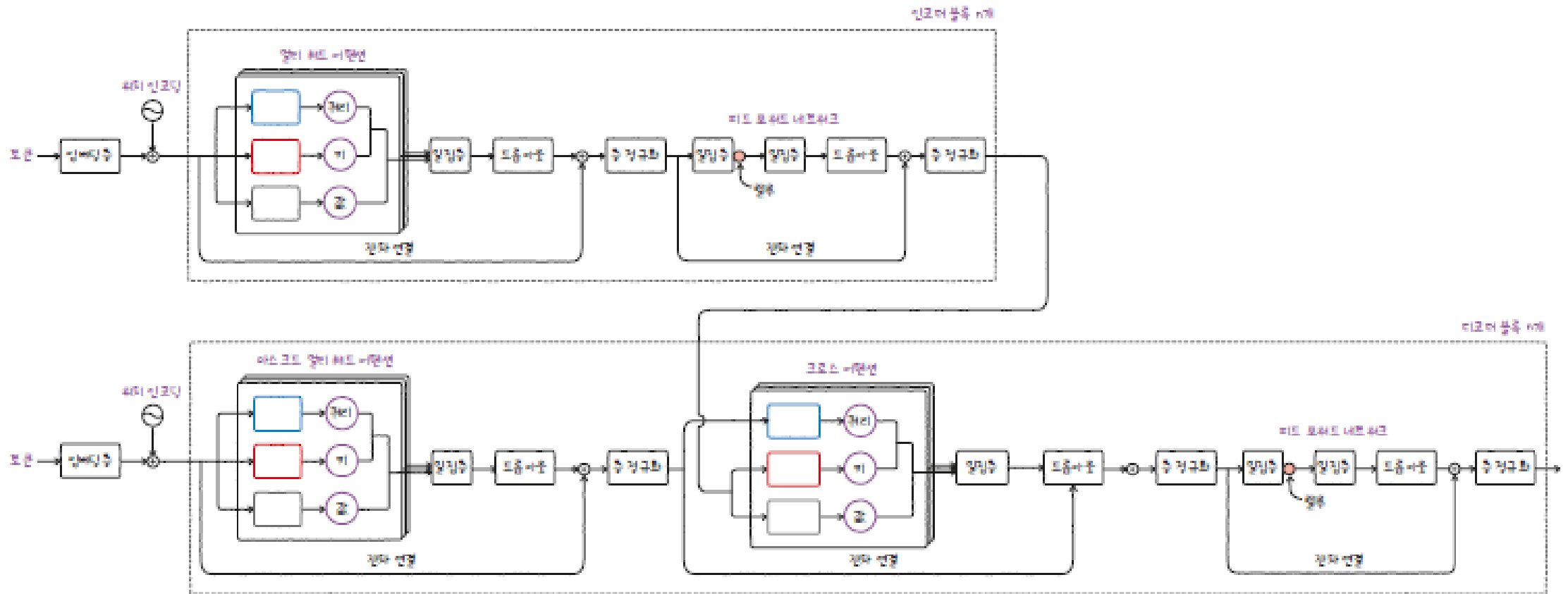
## ○ 디코더 블록

- 하지만 디코더가 다음에 출력할 정답을 미리 알게 되면 올바른 학습이 이루어질 수 없음
  - “나는”이라는 토큰을 처리할 때 “너를 사랑한다”라는 정답을 미리 볼 수 있다면, 모델이 단순히 정답을 복사하는 방식으로 학습될 위험이 있음
- 이를 방지하기 위해 디코더의 첫 번째 멀티 헤드 어텐션 층에서는 마스킹(masking) 처리
  - 즉, 디코더가 한 타임스텝에서 어텐션 점수를 계산할 때 현재 토큰까지만 참고하고, 이후의 토큰은 볼 수 없도록 제한
- 마스크드 멀티 헤드 어텐션(masked multi-head attention) 층

# SECTION 10-1 순차 데이터와 순환 신경망(35)

## ○ 디코더 블록

- 인코더와 디코더를 합친 트랜스포머 전체 모델



# SECTION 10-1 순차 데이터와 순환 신경망(36)

## ○ 디코더 블록

- 트랜스포머는 어텐션, 층 정규화, 잔차 연결, 드롭아웃 같은 기술을 효과적으로 조합하여 새로운 구조를 만들었고, 이를 통해 인코더-디코더 모델에서 순환층을 완전히 제거
- 트랜스포머 모델은 순환 신경망 기반의 인코더-디코더 모델보다 훨씬 높은 성능을 보여주며, 텍스트 처리 분야의 표준 모델로 자리 잡음
- 트랜스포머 모델은 대규모 텍스트 데이터셋을 학습하며, 매우 많은 모델 파라미터를 가지고 있음
  - 대규모 언어 모델(large language model, LLM)
  - 이런 모델을 훈련하는 데는 많은 자원과 비용이 필요

# SECTION 10-1 마무리

## ○ 키워드로 끝나는 핵심 포인트

- 시퀀스-투-시퀀스 작업
  - 시퀀스 데이터를 입력받아 다시 시퀀스 데이터를 출력하는 작업
- 어텐션 메커니즘
  - 인코더-디코더 구조에 사용된 순환 신경망의 성능을 향상
- 트랜스포머 모델
  - 어텐션 메커니즘을 기반으로 한 인코더-디코더 구조에서 순환층을 완전히 제거
- 멀티 헤드 어텐션
  - 트랜스포머 모델의 핵심 구성 요소
  - 어텐션 메커니즘을 계산하는 헤드를 여러 개 병렬로 구성하고 마지막에 밀집층을 두어 원래 임베딩 차원으로 복원하는 구조

## SECTION 10-1 확인 문제(1)

1 트랜스포머 모델에서 인코더의 출력을 표현하는 말이 아닌 것은?

- ① 단어 임베딩
- ② 임베딩 벡터
- ③ 잠재 벡터
- ④ 은닉 벡터

2 트랜스포머 모델의 대표적인 구성 요소가 아닌 것은?

- ① 멀티 헤드 어텐션
- ② 배치 정규화
- ③ 피드포워드 네트워크
- ④ 잔차 연결

## SECTION 10-1 확인 문제(2)

3 원본 트랜스포머 모델이 단어 임베딩에 위치 정보를 주입하기 위해 사용한 함수는?

- ① 사인 함수와 코사인 함수
- ② 시그모이드 함수
- ③ 렐루 함수
- ④ 하이퍼볼릭 탄젠트 함수

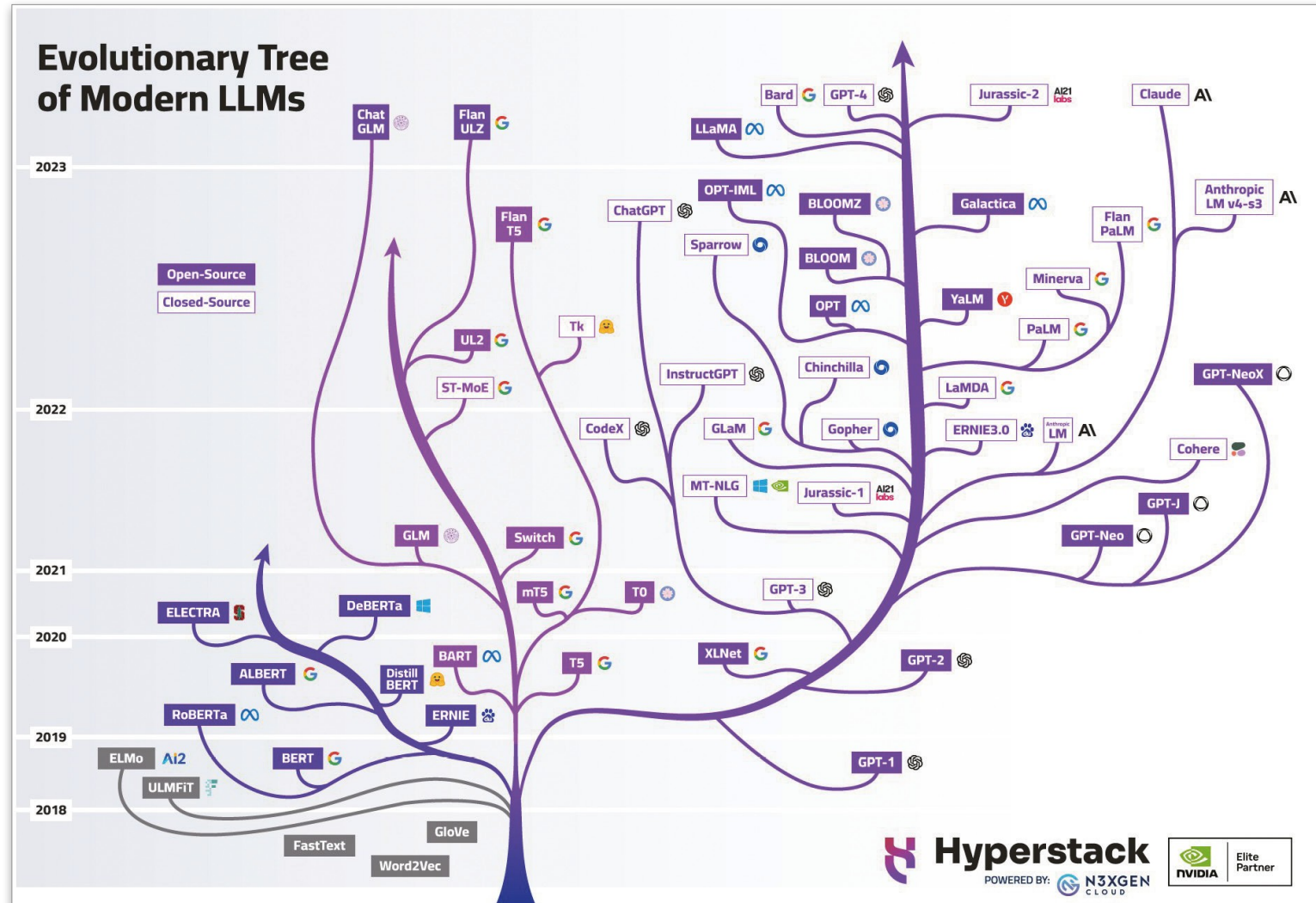
# SECTION 10-2 트랜스포머로 상품 설명 요약하기(1)

## ○ 트랜스포머 가계도

- 트랜스포머 모델은 인코더와 디코더로 구성
  - 인코더 또는 디코더만 따로 떼어내어 사용할 수도 있음
    - 인코더, 디코더, 인코더-디코더 구조에서 출력하는 결과는 모두 동일한 형태의 은닉 벡터
    - 세 구조의 출력 형태가 같기 때문에, 특정 작업에 따라 적절한 구조를 선택할 수 있음
  - 인코더-디코더 모델
    - 요약과 번역 같은 전형적인 시퀀스-투-시퀀스 작업
    - 질문-답변(question answering, QA)
  - 인코더 기반 모델
    - 개체명 인식(named entity) - 텍스트에서 사람 이름, 지역명, 회사 이름 등의 고유 명사를 식별하는 작업
      - 이 경우 입력된 각 토큰마다 개체명 여부를 출력
    - STS(semantic textual similarity) - 두 텍스트의 유사도를 측정
  - 디코더 기반 모델

# SECTION 10-2 트랜스포머로 상품 설명 요약하기(2)

## ○ 트랜스포머 가계도





## SECTION 10-2 트랜스포머로 상품 설명 요약하기(3)

### ○ 트랜스포머 가계도 (설명)

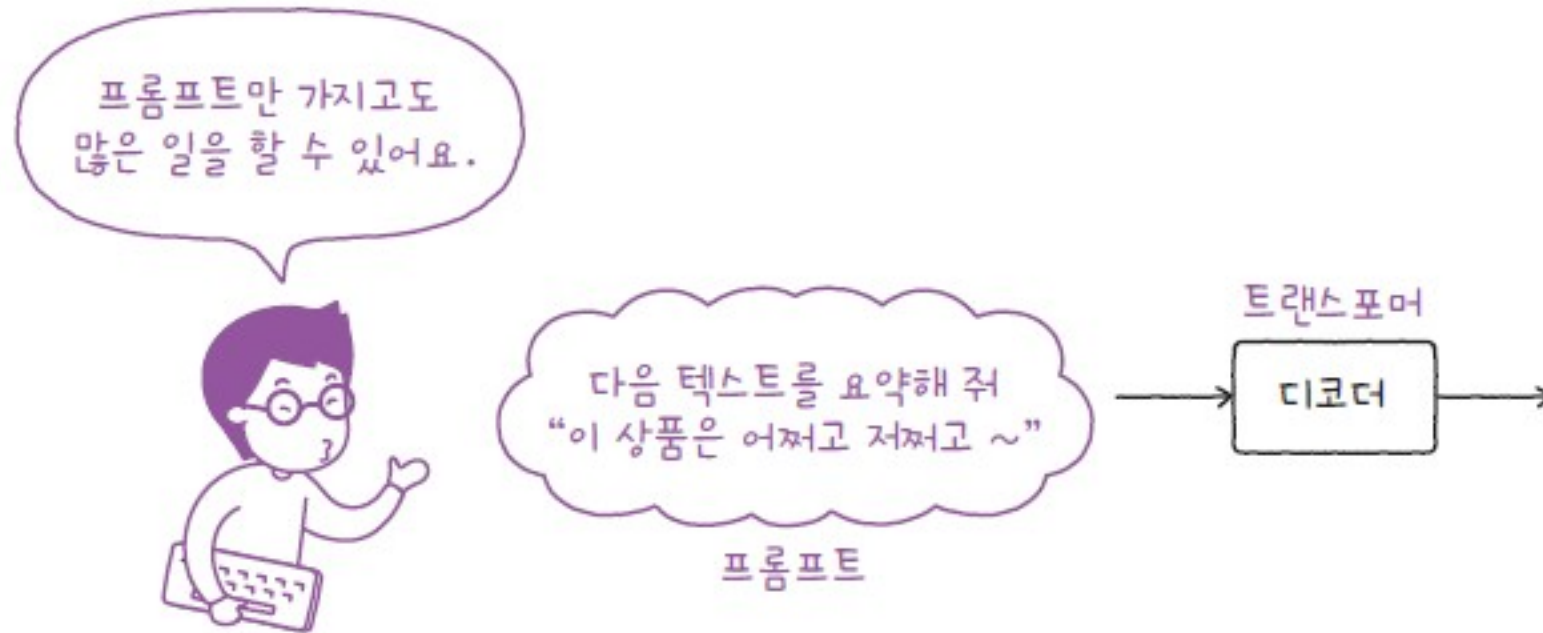
- 2018년 즈음부터 언어 모델의 주요 발전 과정
- 가장 왼쪽에 회색 부분
  - 유용한 단어 임베딩 벡터를 만드는 데 초점을 맞춘 모델들
  - 이 모델들은 트랜스포머 구조를 사용하지 않으며, 초창기 LLM
- 중앙에 크게 세 개의 가지가 위로 뻗어 있는 부분
  - 트랜스포머 구조를 사용
  - 가장 왼쪽의 진보라색 가지는 인코더 기반 모델 - 이 가지는 2021년 후에 더 이상 자라지 못하고 있음  
실제로 인코더 기반 모델에 대한 관심이 줄었기 때문임
    - 대표적인 인코더 기반 모델은 BERT, RoBERT, ELECTRA 등
  - 가운데 연보라색 가지는 인코더-디코더 모델
    - 대표적인 인코더-디코더 모델은 T5, BART 등
- 가장 오른쪽에 있는 보라색 가지
  - 디코더 기반 모델
  - GPT-3와 ChatGPT 같은 유명한 모델이 여기 포함
  - 보라색 바탕 - 오픈소스 모델
  - 흰색 바탕 - 클로즈드 소스, 즉 독점적인 상업 모델

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(4)

### ○ 트랜스포머 가계도

#### - 디코더 기반 모델이 특히 인기가 높은 이유

- 뛰어난 텍스트 생성 능력 - 텍스트 생성 기능은 챗봇, 질문 답변, 요약, 번역 등에 널리 적용
- 디코더는 이전까지 생성한 텍스트를 입력으로 받아 다음 토큰을 예측하는 식으로 동작
  - 프롬프트(prompt) - 이전에 생성한 텍스트인 것처럼 전달하는 초기 텍스트 - 사람이 모델을 실행할 때 전달



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(5)

### ○ 트랜스포머 가계도

#### - 오픈 소스와 클로즈드 소스

- 초기 대규모 언어 모델은 대부분의 머신러닝 분야가 그렇듯이 오픈소스
- 디코더 기반 모델 중에서 GPT-3가 크게 성공하자 클로즈드 소스 정책으로 선회했고 ChatGPT 같은 유료 서비스가 등장
- 이후로 많은 회사들이 디코더 기반의 모델들을 클로즈드 소스로 출시
- 2022년부터 오픈소스 모델들이 다시 등장 - 클로즈드 소스 LLM과 경쟁하기 시작
  - 메타(Meta)에서 공개한 Llama와 Llama에서 파생된 많은 모델이 2023년을 언어 모델의 해로 만들었음
- 2025년에도 (그리고 아마 그 이후에도) 이런 추세는 계속
  - 구글의 오픈소스 LLM인 Gemma
  - 마이크로소프트 오픈소스 Phi 모델
  - 알리바바 그룹의 Qwen
- 오픈소스 LLM이 인기가 높은 것은 단순히 구조만 공개된 것이 아니라 대규모 텍스트 말뭉치에서 훈련된 모델 파라미터도 함께 공개했기 때문임

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(6)

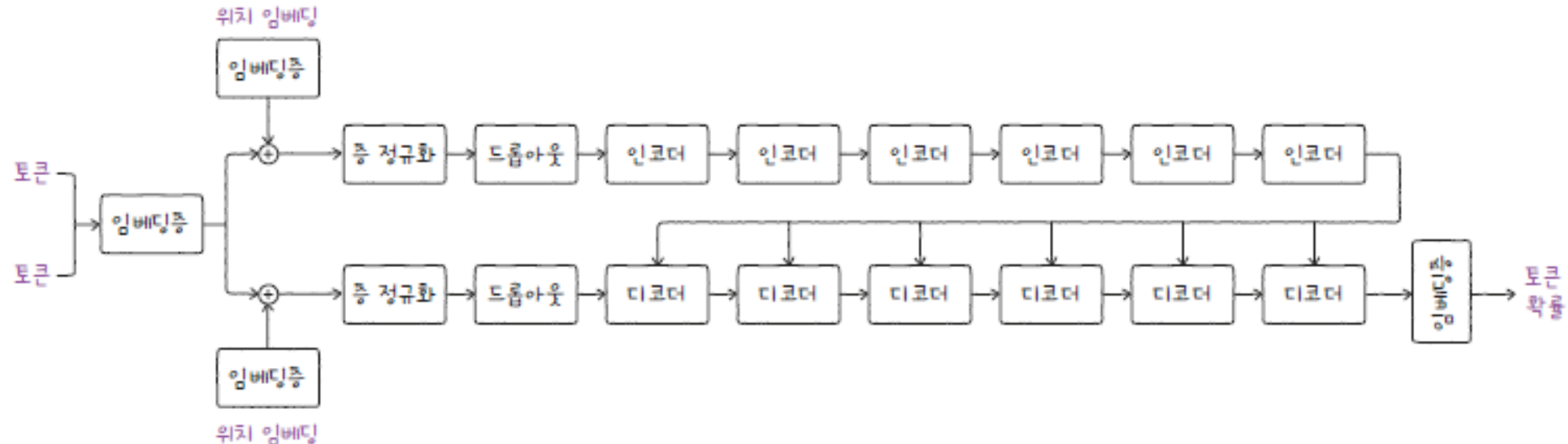
### ○ 전이 학습

- 전이 학습(transfer learning)
  - 이미 훈련된 모델을 새로운 작업에 맞춰 재사용하거나 약간 조정하여 사용하는 방법
- 이미지넷 데이터셋(ImageNet dataset)
  - 스탠포드 대학교에서 만든 컴퓨터 비전을 위한 대규모 이미지 데이터베이스
  - 훈련된 신경망이 많이 공개되어 있으며, 이런 신경망은 구조와 가중치가 모두 공개되어 있기 때문에 누구나 자신의 작업에 가져다 사용할 수 있음
- 베이스 모델(base model)
  - 일반적으로 합성곱 신경망의 마지막 부분에 놓인 한 개 이상의 밀집층(또는 분류층)을 버리고 입력부터 마지막 합성곱 층까지만 재사용
- 트랜스포머 기반 언어 모델이 인기를 끄는 가장 큰 이유는 전이 학습이 가능하기 때문임

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(7)

### ○ BART 모델 소개

- 2019년 메타에서 공개한 트랜스포머 기반의 인코더-디코더 언어 모델
  - 160GB에 달하는 대규모 텍스트 데이터셋으로 훈련
- 베이스(base) 모델
  - 인코더와 디코더 블록을 각각 6개씩 사용 / 파라미터 개수는 약 1억 2천만 개
- 라지(large) 모델
  - 인코더와 디코더 블록을 각각 12개씩 사용 / 파라미터 개수는 4억 개 이상



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(8)

### ○ BART 모델 소개

#### - 위치 임베딩(positional embedding)

- 토큰 아이디를 실수 벡터 표현으로 바꾸기 위해 임베딩 층을 사용하듯 위치 정숫값을 실수 벡터 표현으로 바꾸기 위해 임베딩 층을 사용
- 토큰을 위한 임베딩 층처럼 위치를 위한 임베딩 층도 처음에는 랜덤하게 초기화되며 훈련을 통해 최적의 값을 학습
- 트랜스포머 기반 인코더-디코더 모델은 하나의 공통된 단어 임베딩 층을 사용하여 인코더와 디코더가 동일한 방식으로 토큰을 변환하도록 설계

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(9)

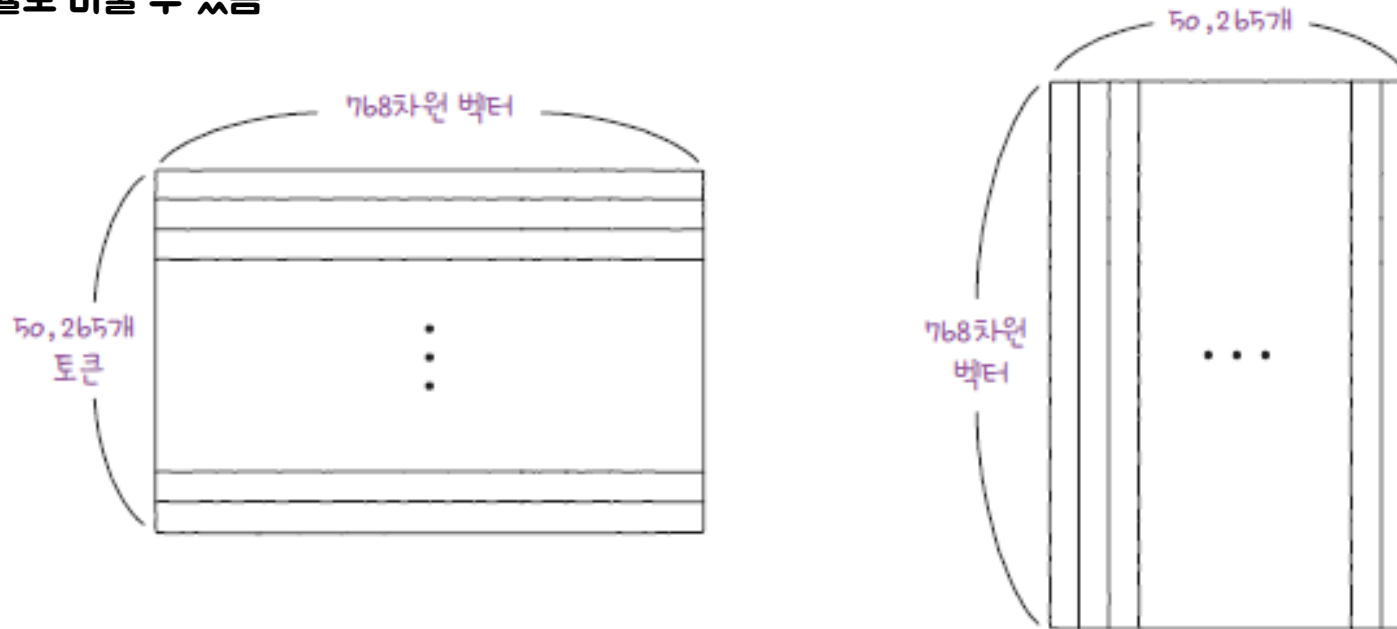
### ○ BART 모델 소개

- 토큰 임베딩과 위치 임베딩을 더한 후 층 정규화와 드롭아웃 층을 거치고, 그다음 인코더 블록과 디코더 블록을 반복해서 거침
- 디코더 마지막에 세로로 회전시킨 임베딩 층을 통과하면 토큰에 대한 확률이 출력
  - 디코더의 마지막 출력은 토큰에 대한 은닉 벡터 즉, 임베딩 벡터
  - BART 베이스 모델의 경우 이 벡터의 크기는 768
    - 이 벡터로부터 다음 토큰에 대한 확률을 계산
  - BART 모델이 출력할 수 있는 어휘 사전의 토큰 수는 50,265개
    - 768 크기의 벡터를 50,265개의 확률을 담은 또 다른 벡터로 바꾸는 작업이 필요
    - 입력이 768개일 때 50,265개의 출력을 만드는 밀집층으로 쉽게 구현
    - 이 밀집층에는  $768 \times 50,265$ 개의 모델 파라미터가 필요

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(10)

### ○ BART 모델 소개

- 베이스 모델에는 이와 동일한 크기의 모델 파라미터가 모델 맨 처음에 이미 사용되고 있음
  - 토큰 정수를 단어 임베딩으로 만드는 임베딩 층
  - 이 층에는 50,265개의 토큰에 대한 단어 임베딩이 학습되어 있음
  - 각 단어 임베딩 벡터의 차원은 768
- 이 행렬을 살짝 회전시킨다면 다음처럼  $768 \times 50,265$  크기의 모델 파라미터를 얻을 수 있음
  - 별도의 밀집층을 두지 않고 임베딩 층의 모델 파라미터를 사용해 디코더의 출력 벡터를 각 토큰에 대한 확률로 바꿀 수 있음

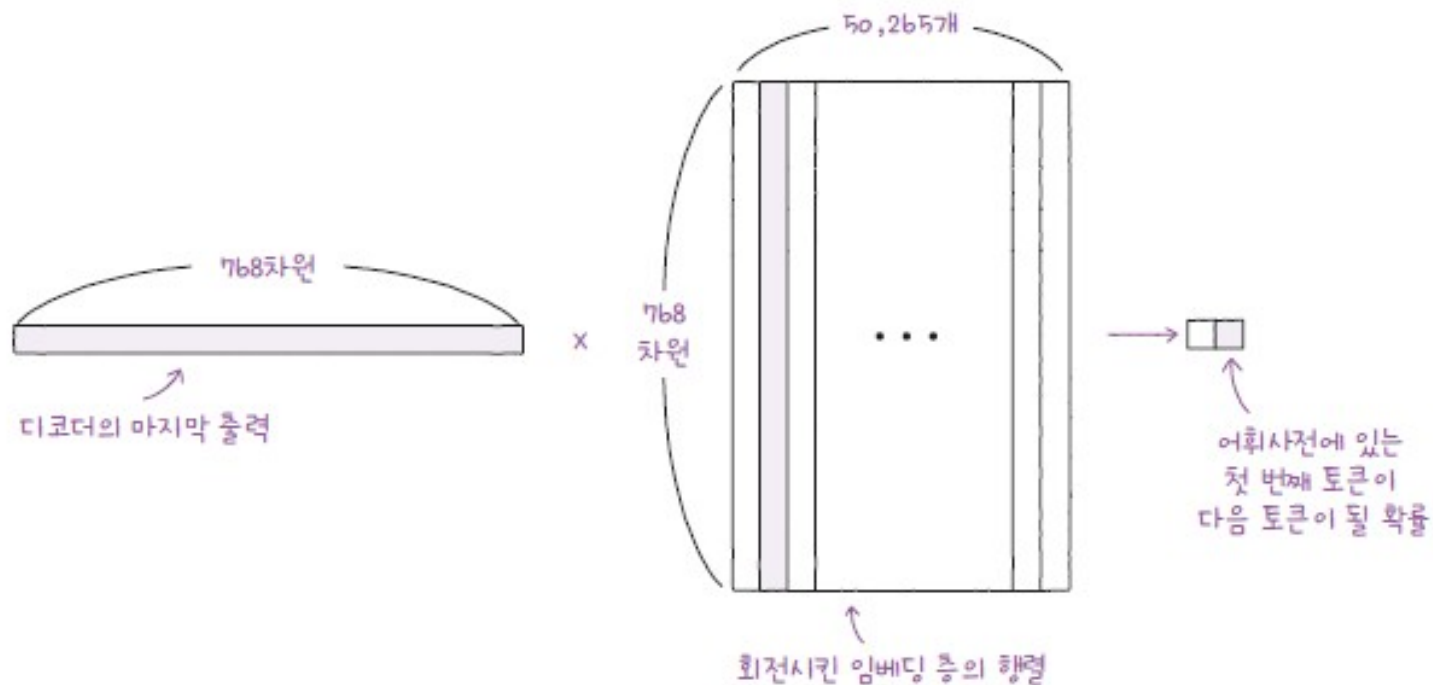




## SECTION 10-2 트랜스포머로 상품 설명 요약하기(11)

### ○ BART 모델 소개

- 대규모 언어 모델을 만들 때 종종 적용되며 마지막에 밀집층을 따로 둘 필요가 없어 전체 모델 파라미터의 수를 줄이는 데 도움이 됨
- 1) 디코더 층의 마지막 출력을 회전시킨 임베딩 행렬의 첫 번째 열과 곱하면 하나의 실숫값을 얻을 수 있음
  - 이 값을 어휘 사전에 있는 첫 번째 토큰이 선택될 확률로 생각할 수 있음

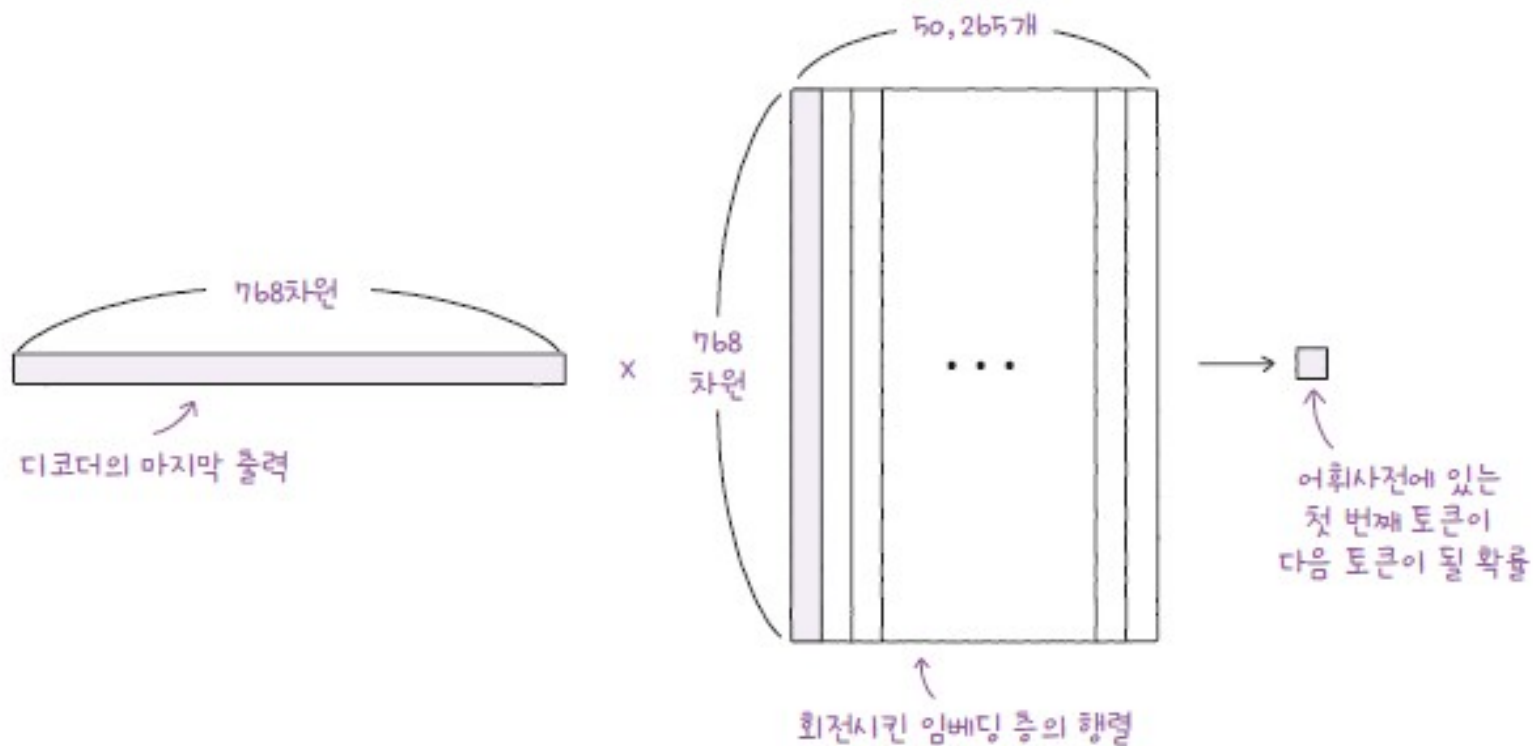


## SECTION 10-2 트랜스포머로 상품 설명 요약하기(12)

### ○ BART 모델 소개

2) 그다음 두 번째 열과 곱하면 다음과 같이 두 번째 토큰이 다음 토큰으로 선택될 확률을 출력

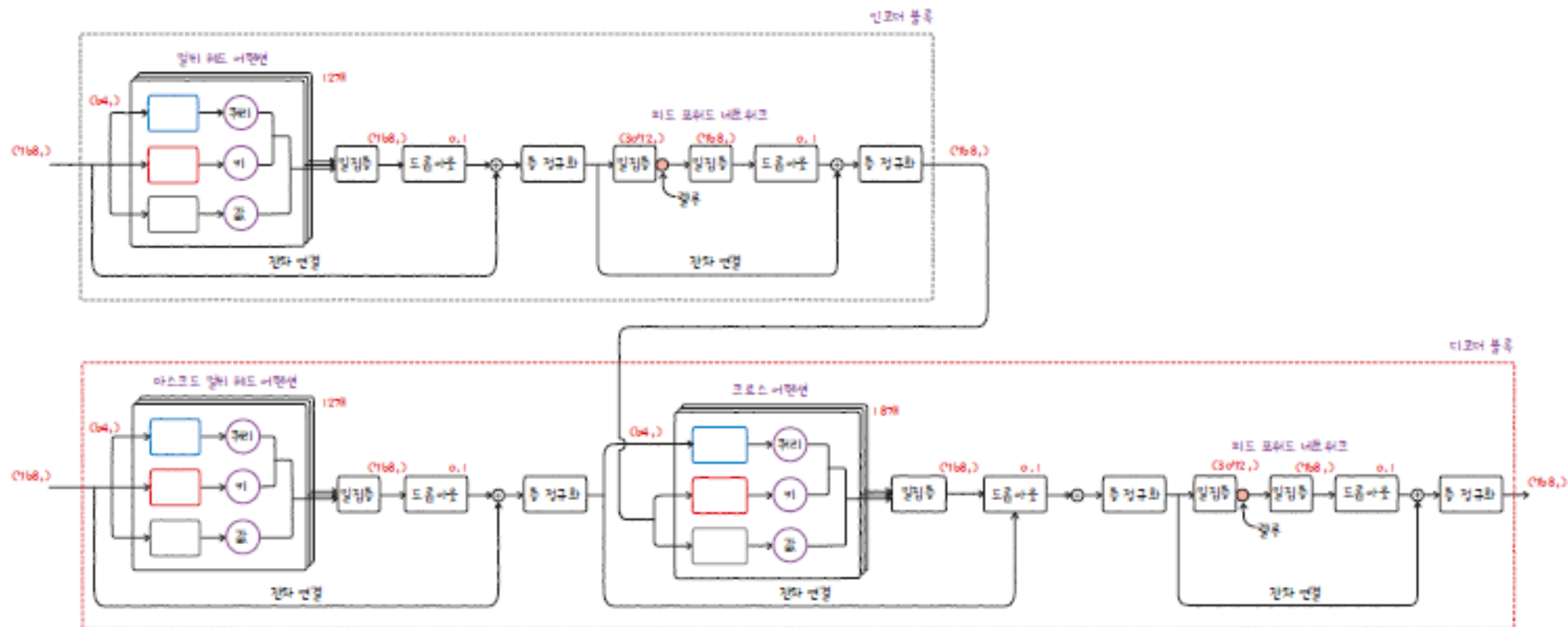
- 50,265개의 열과 모두 곱하면 50,265개의 실숫값이 만들어지고 이를 다음 토큰에 대한 확률처럼 생각할 수 있음



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(13)

### ○ BART의 인코더와 디코더

- BART의 인코더 블록과 디코더 블록은 원본 트랜스포머의 구조와 매우 유사함
  - (다른 점) 피드포워드 네트워크에서 렐루 활성화 함수 대신 젤루GeLU 함수를 사용



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(14)

### ○ BART의 인코더와 디코더

#### - BART의 인코더 블록(앞의 그림 설명)

- 인코더 입력 - (768,) 크기의 임베딩 벡터가 인코더 블록에 입력되면, 12개의 헤드에 나누어 전달
  - 따라서 각 헤드에 입력되는 벡터 크기는 (64,)
  - 멀티 헤드 어텐션의 마지막 밀집층을 통과하면서 다시 (768,) 크기의 벡터로 변환
- BART는 인코더와 디코더에 드롭아웃 비율이 0.1인 드롭아웃 층을 사용
  - 드롭아웃 층을 지난 후 잔차 연결이 나오고 층 정규화가 이어짐
- 피드포워드 네트워크
  - 첫 번째 밀집층은 임베딩 벡터 차원의 네 배인 (3072,)의 출력을 만듦
  - 두 번째 밀집층은 다시 원래 임베딩 차원인 (768,)의 출력
- 그다음 앞서와 동일하게 드롭아웃, 잔차 연결, 층 정규화가 등장
  - 이 세 층은 벡터의 차원을 바꾸지 않으므로 인코더 블록의 최종 출력 크기는 (768,)

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(15)

### ○ BART의 인코더와 디코더

- BART의 피드포워드 네트워크에서 사용하는 활성화 함수는 젤루
  - 입력에 표준 정규 분포의 누적 분포 함수(cumulative distribution function for gaussian distribution)를 곱함
  - 이 누적 분포 함수를 계산하려면 적분이 필요
    - 그래서 대부분의 딥러닝 프레임워크들은 복잡한 적분 대신 근사값을 구할 수 있는 간단한 공식을 사용

젤루      입력

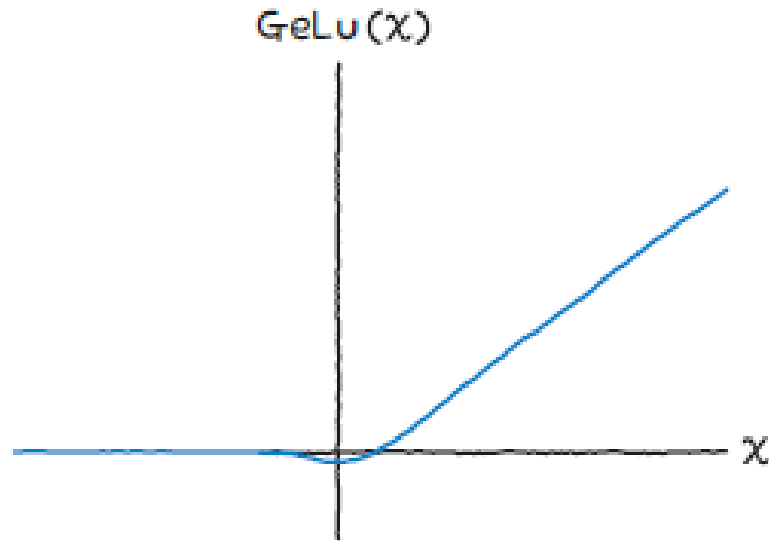
$$\text{GeLU}(x) = x \cdot \Phi(x) = \frac{1}{2} x \left( 1 + \tanh\left(\sqrt{\frac{2}{\pi}} (x + 0.044715 x^3)\right) \right)$$

표준 정규 분포의  
누적 분포 함수

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(16)

### ○ BART의 인코더와 디코더

- 젤루 함수의 그래프는 다음처럼 렐루 함수와 비슷하지만 원점에서 부드럽게 변하기 때문에 미분 가능



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(17)

### ○ BART의 인코더와 디코더

#### - BART의 디코더 블록(앞의 그림 설명)

- 인코더 블록과 매우 흡사한 처리 과정
- 맨 처음 (768,) 크기의 입력이 마스크드 멀티 헤드 어텐션 층에 12개의 헤드에 나뉘어져 들어감
- 어텐션 바로 뒤에 등장하는 밀집층이 출력 차원을 (768,)로 만들고, 드롭아웃, 잔차 연결, 층 정규화가 등장
- 두 번째 멀티 헤드 어텐션은 인코더의 출력을 사용하는 크로스 어텐션
- 인코더의 출력은 키와 값으로 전달되고, 디코더의 벡터를 쿼리로 사용하여 어텐션을 계산
- 그다음은 역시 동일하게 드롭아웃, 잔차 연결, 층 정규화가 등장
- 디코더의 출력 크기도 입력과 동일하게(768,)

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(18)

### ○ 허깅페이스로 KoBART 모델 로드하기

#### - 허깅페이스(HuggingFace)의 transformers 패키지

- 허깅페이스는 트랜스포머 기반의 오픈소스 모델을 공유
- transformers 패키지를 사용하면 허깅페이스에서 공유되는 사전 훈련된 오픈소스 LLM을 쉽게 로드하여 사용할 수 있고, 직접 미세 튜닝한 모델도 허깅페이스를 통해 공유할 수 있음
- 허깅페이스는 LLM의 개발에 필요한 다양한 패키지를 계속 개발하여 공개
- 또한 모델뿐만 아니라 데이터셋과 교육 자료도 풍부
- 트랜스포머 기반 LLM 모델을 넘어서 컴퓨터 비전과 오디오 등 다른 분야의 모델도 제공



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(19)

### ○ 허깅페이스로 KoBART 모델 로드하기

- 구글 코랩의 transformers 패키지에서 pipeline ( ) 함수를 로드하여 텍스트 요약을 위한 파이프라인을 준비
  - 만약 로컬 컴퓨터에서 코드를 실행한다면 다음 명령으로 transformers 패키지를 먼저 설치

```
!pip install transformers
```

```
from transformers import pipeline
```

```
pipe = pipeline(task='summarization', device=0)
```

No model was supplied, defaulted to sshleifer/distilbart-cnn-12-6 and revision a4f8f3e (<https://huggingface.co/sshleifer/distilbart-cnn-12-6>). Using a pipeline without specifying a model name and revision in production is not recommended.



config.json: 100%	<div></div>	1.80k/1.80k [00:00<00:00, 111kB/s]
pytorch_model.bin: 100%	<div></div>	1.22G/1.22G [00:11<00:00, 184MB/s]
tokenizer_config.json: 100%	<div></div>	26.0/26.0 [00:00<00:00, 1.72kB/s]
vocab.json: 100%	<div></div>	899k/899k [00:00<00:00, 7.01MB/s]
merges.txt: 100%	<div></div>	456k/456k [00:00<00:00, 9.75MB/s]

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(20)

### ○ 허깅페이스로 KoBART 모델 로드하기

#### - pipeline ( ) 함수

- 허깅페이스에 있는 LLM을 사용하기 위해 수행할 몇 가지 단계를 한 번에 실행할 수 있어 매우 편리
- 함수의 매개변수
  - task - 기본이 되는 매개변수, 수행할 작업의 종류를 지정  
앞의 코드에서는 요약 작업을 위해 'summarization'으로 지정  
task 매개변수에 지정할 수 있는 옵션  
    텍스트 분류를 위한 'text-classification'  
    텍스트 생성을 위한 'text-generation'  
    번역을 위한 'translation' 등
  - device - GPU 사용  
이 장에서는 코랩의 T4 GPU 하나를 사용한다고 가정하므로 device=0으로 지정

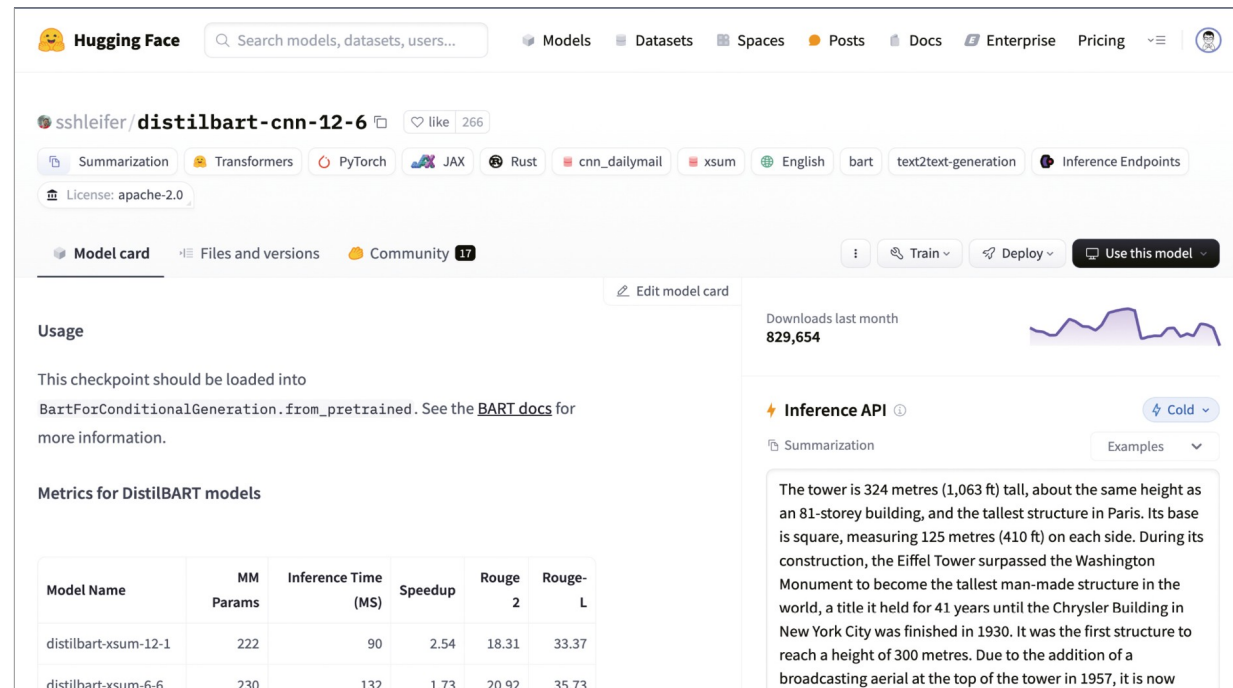
```
pipe = pipeline(task='summarization',  
                model='sshleifer/distilbart-cnn-12-6', device=0)
```

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(21)

### ○ 허깅페이스로 KoBART 모델 로드하기

#### - 모델 이름은 허깅페이스 웹사이트의 경로를 나타냄

- 따라서 다음처럼 <https://huggingface.co/sshleifer/distilbart-cnn-12-6>에 접속하면 이 모델에 대한 상세 내용을 확인
- 반대로 허깅페이스 웹사이트에서 찾은 어떤 모델을 pipeline ( ) 함수로 로드하고 싶다면 URL에서 <https://huggingface.co/> 다음에 나오는 경로를 model 매개변수에 지정



**Hugging Face** Search models, datasets, users... Models Datasets Spaces Posts Docs Enterprise Pricing

sshleifer/**distilbart-cnn-12-6** like 266

Summarization Transformers PyTorch JAX Rust cnn\_dailymail xsum English bart text2text-generation Inference Endpoints

License: apache-2.0

Model card Files and versions Community 17

**Usage**

This checkpoint should be loaded into `BartForConditionalGeneration.from_pretrained`. See the [BART docs](#) for more information.

**Metrics for DistilBART models**

Model Name	MM Params	Inference Time (MS)	Speedup	Rouge 2	Rouge-L
distilbart-xsum-12-1	222	90	2.54	18.31	33.37
distilbart-xsum-6-6	230	132	1.73	20.92	35.73

Downloads last month: 829,654

**Inference API** Cold

Summarization Examples

The tower is 324 metres (1,063 ft) tall, about the same height as an 81-storey building, and the tallest structure in Paris. Its base is square, measuring 125 metres (410 ft) on each side. During its construction, the Eiffel Tower surpassed the Washington Monument to become the tallest man-made structure in the world, a title it held for 41 years until the Chrysler Building in New York City was finished in 1930. It was the first structure to reach a height of 300 metres. Due to the addition of a broadcasting aerial at the top of the tower in 1957, it is now

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(22)

### ○ 허깅페이스로 KoBART 모델 로드하기

- pipe 객체로 텍스트를 요약하려면 8장에서 해보았던 것처럼 이 객체를 마치 함수처럼 호출
  - 예) 반 고흐에 관한 위키백과 텍스트를 요약
  - BART 모델은 CNN 텍스트 데이터셋에서 미세 튜닝된 모델
  - 이 모델은 텍스트를 56~142자 사이의 길이로 요약  
만약 이 두 값의 범위를 바꾸고 싶다면 각각 min\_length와 max\_length 매개변수를 사용

```
sample_text = """Vincent Willem van Gogh was a Dutch Post-Impressionist painter who is among the most famous and influential figures in the history of Western art. In just over a decade, he created approximately 2100 artworks, including around 860 oil paintings, most of them in the last two years of his life. His oeuvre includes landscapes, still lifes, portraits, ...  
Van Gogh's paintings, The Red Vineyard, was sold.  
"""  
pipe(sample_text)
```

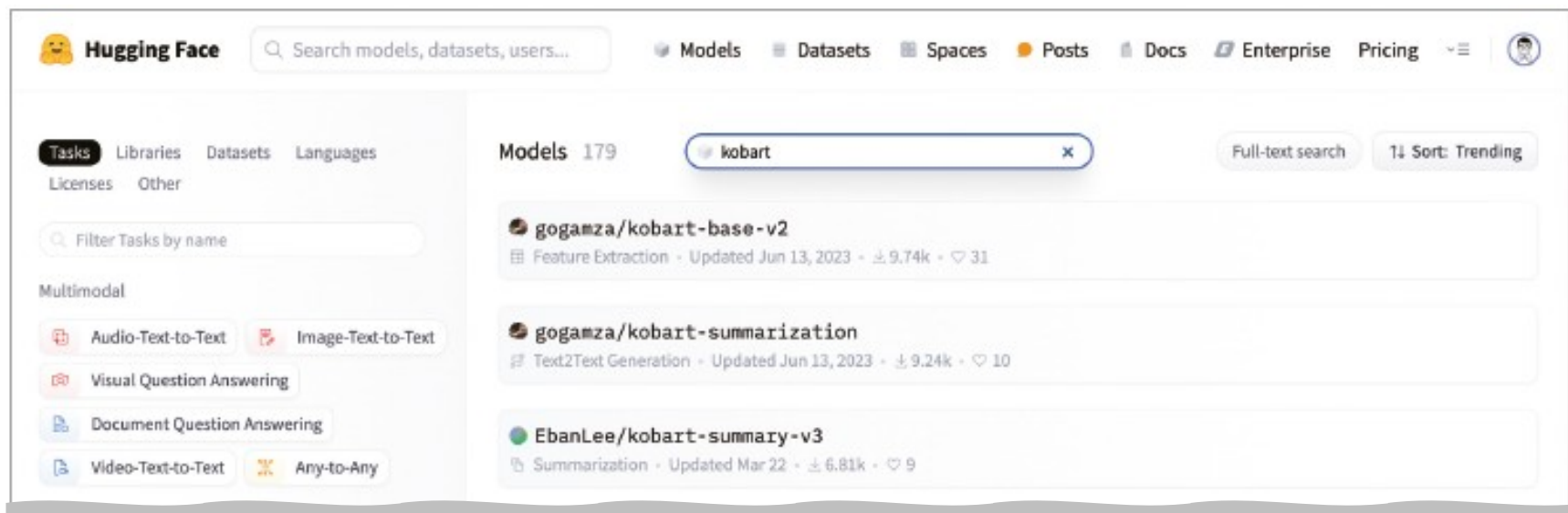


```
[{'summary_text': " Vincent Willem van Gogh was a Dutch Post-Impressionist painter . His oeuvre includes landscapes, still lifes, portraits and selfportraits . Van Gogh's work was beginning to gain critical attention before he died from a self-inflicted gunshot at age 37 ."}]
```

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(23)

### ○ 허깅페이스로 KoBART 모델 로드하기

- sshleifer/distilbart-cnn-12-6 모델은 영어 데이터셋에서 훈련되었기 때문에 한글과 같은 다국어어를 지원하지 못함
- BART 베이스 모델을 사용해 한글 데이터셋에서 미세 튜닝한 다른 모델을 허깅페이스에서 찾기
  - 허깅페이스 웹사이트에서 맨 위쪽 Models를 클릭(<https://huggingface.co/models>)
  - 검색어로 kobart를 입력
  - KoBART는 SKT에서 만든 BART기반의 한국어 인코더-디코더 모델(<https://github.com/SKT-AI/KoBART>)



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(24)

### ○ 허깅페이스로 KoBART 모델 로드하기

- KoBART를 요약 작업에 맞춰 미세 튜닝한 모델 중 비교적 최근에 업데이트된 EbanLee/kobart-summary-v3 모델을 사용

```
kobart = pipeline(task='summarization',  
                  model='EbanLee/kobart-summary-v3', device=0)
```

- 요약할 텍스트로는 『혼자 공부하는 데이터 분석 with 파이썬』의 도서 소개를 사용

```
ko_text = """  
하나, '입문자 맞춤형 7단계 구성'을 따라가며 체계적으로 반복하는 탄탄한 학습 설계!  
이 책은 데이터 분석의 핵심 내용을 7단계에 걸쳐 반복 학습하면서 자연스럽게 머릿속에 기억되  
도록 구성했습니다. [핵심 키워드]와 [시작하기 전에]에서 각 절의 주제에 대한 대표 개념을 위  
밍업하고, 이론과 실습을 거쳐 마무리에서는 [핵심 포인트]와 [확인 문제]로 한번에 복습합니다.  
"""  
kobart(ko_text)
```

['summary\_text': '이 책은 데이터 분석의 핵심 내용을 7단계에 걸쳐 반복 학습하면서 머릿속에 기억되도록 구성했습니다. 독자 공부할 수 있는 커리큘럼을 그대로 믿고 끝까지 따라가다 보면 데이터 분석 공부가 난생 처음인 입문자도 무리 없이 책을 끝까지 마칠 수 있습니다. 현장감 넘치는 스토리를 통해 데이터를 다루는 방법을 알려 주어 몰입감 있는 학습을 할 수 있도록 구성했습니다. 저자가 질문 하나하나에 직접 답변을 달아 주는 것은 물론, 최신 기술과 정보도 얻을 수 있습니다. 혼공 학습단과 함께하면 마지막까지 포기하지 않고 완주할 수 있을 것입니다.』

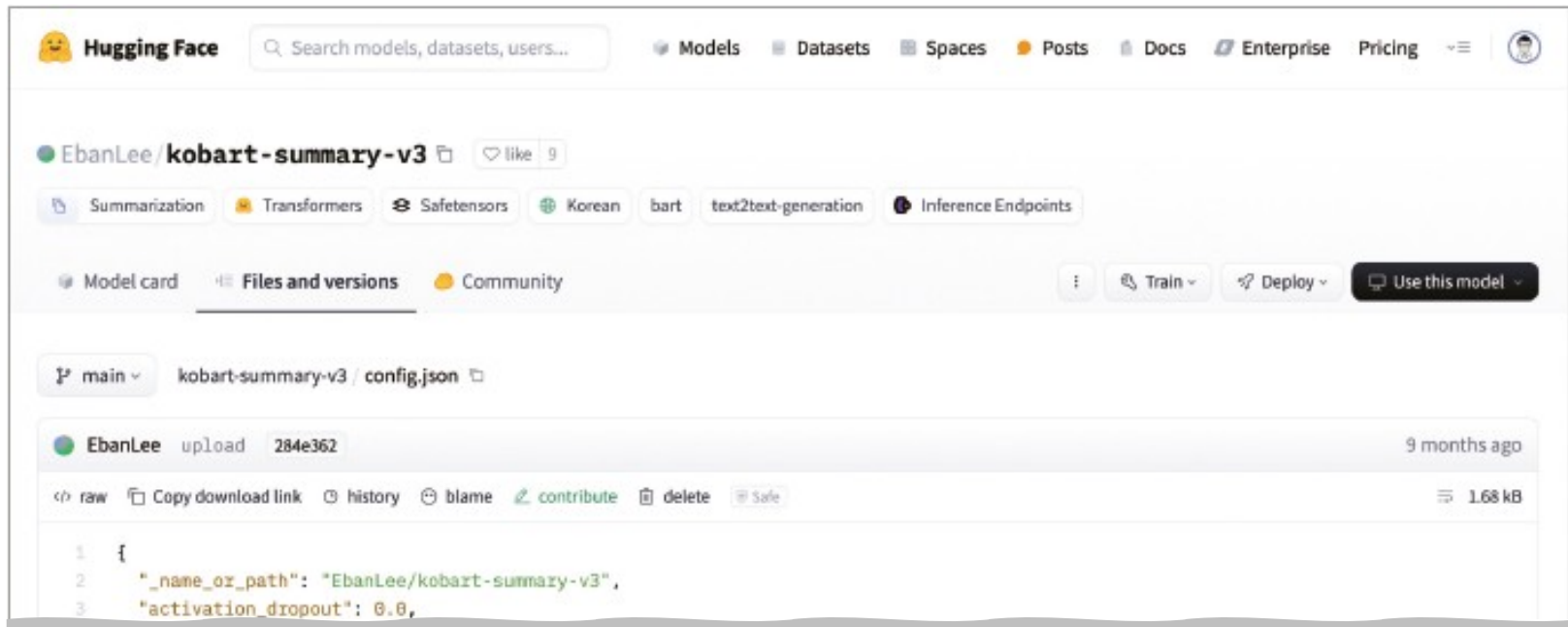
빔 서치(beam search)라는 방법을 사용해 텍스트를 생성하기 때문에 실행할 때마다 결과가 다를 수 있음

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(25)

+ 여기서 잠깐

모델이 최대 300자까지 출력한다는 것을 어떻게 아나요? 소스 코드를 봐야 하나요?

- 허깅페이스의 모델은 웹 사이트에서 자세한 설정을 확인할 수 있음
- EbanLee/kobart-summary-v3 모델 페이지(<https://huggingface.co/EbanLee/kobart-summary-v3>)에 접속한 다음 Files 탭을 선택
- 파일 목록에서 config.json을 클릭하면 아래 그림처럼 자세한 모델 설정을 볼 수 있음



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(26)

+ 여기서 잠깐

모델이 최대 300자까지 출력한다는 것을 어떻게 아나요? 소스 코드를 봐야 하나요?

- 또는 모델 객체의 config 속성을 확인
- 파이프라인 함수로 로드한 모델은 파이프라인 객체인 kobart의 model 속성에 저장
- 이 모델의 설정을 확인하려면 다음처럼 kobart.model.config를 출력
  - 출력된 내용에서 max\_length의 값을 확인

```
print(kobart.model.config)
BartConfig {
  "_attn_implementation_autoset": true,
  "_name_or_path": "ebanlee/kobart-summary-v3",
  "activation_dropout": 0.0,
  ...
  "task_specific_params": {
    "summarization": {
      "length_penalty": 1.0,
      "max_length": 300,
      "min_length": 12,
      "no_repeat_ngram_size": 15,
      "num_beams": 6,
      "repetition_penalty": 1.5
    }
  }
}
```

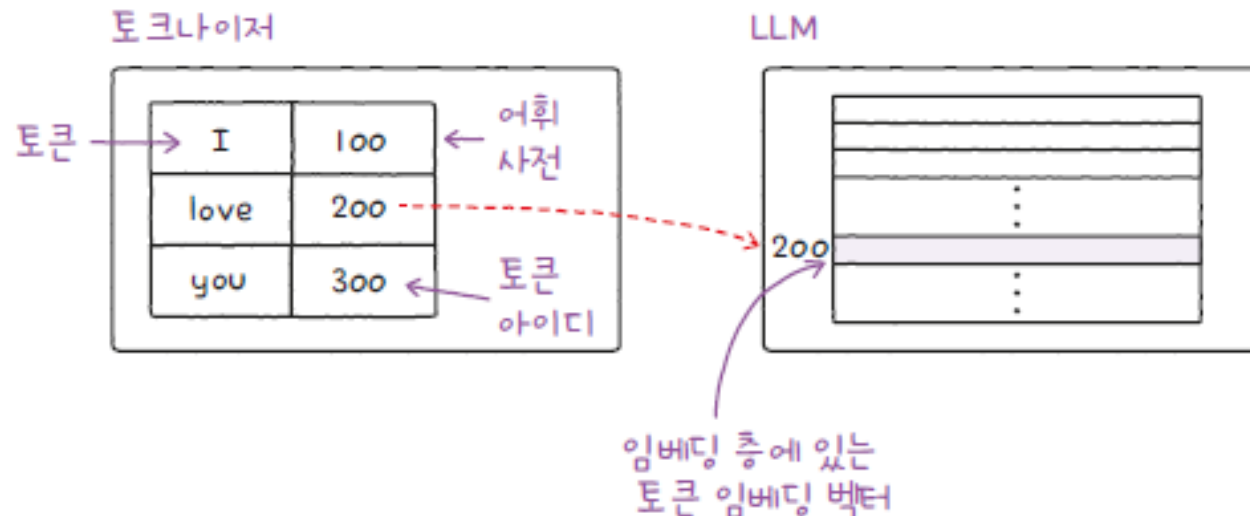
(이하 생략)



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(27)

### ○ 텍스트 토큰화

- 토큰화(tokenization) - 텍스트를 토큰(token)이라는 단위로 분할하는 과정
  - 이런 토큰화를 LLM 모델 자체가 수행하지 않음
    - 이 모델에는 텍스트를 토큰으로 분할하는 구성 요소가 없음
    - 이미 텍스트가 토큰으로 분할되고 각 토큰에 정수 아이디가 할당된 후 이 정수 리스트가 전달된다고 가정
- 토크나이저(tokenizer) - 토큰화를 수행하는 모델이며 모델과 별도로 존재
  - 토크나이저는 훈련 데이터로부터 최적의 어휘 사전을 학습하는 모델에 가까움
  - 토큰의 임베딩 벡터는 토크나이저에 있지 않고 LLM 모델의 임베딩 층에 있음



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(28)

### ○ 텍스트 토큰화

- 부분단어(subword) 토큰화 방식 - 트랜스포머 기반 LLM에서 널리 사용
- 대표적인 부분단어 토큰화 방법
  - BPE(Byte-Pair Encoding)
  - 워드피스(WordPiece)
  - 유니그램(unigram)
  - 센텐스피스(SentencePiece)

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(29)

### ○ BPE

- BPE는 먼저 각 단어를 문자 단위로 분해하여 어휘 사전에 추가한 후, 가장 많이 등장하는 순서대로 문자 쌍(또는 부분단어 쌍)을 찾아 병합하고, 합쳐진 부분단어를 어휘 사전에 추가
  - 이 과정을 사전에 정의된 어휘 사전 크기에 도달할 때까지 계속 진행
- BPE 알고리즘의 단점은 유니코드 문자를 처리할 때 발생
  - 이 글을 쓰는 시점에서 유니코드 문자의 개수는 154,998개
    - 따라서 최악의 경우 어휘 사전의 크기가 154,998개부터 시작하게 됨
    - 앞서 살펴본 BART 모델의 어휘 사전 크기가 50,265개였던 것을 생각하면 매우 큰 값
  - 따라서 적절한 수준의 어휘 사전 크기에서 유니코드를 처리할 수 있는 방법이 필요
  - 바이트 수준의 BPE(byte-level BPE) 알고리즘
    - 텍스트를 바이트 스트림으로 인식하고 자주 등장하는 바이트 쌍을 어휘사전에 추가
    - 이렇게 하면 유니코드 문자도 바이트 수준에서 병합하여 어휘사전에 추가할 수 있음

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(30)

### ○ BPE

#### - 앞에서 로드한 kobart 모델의 어휘사전을 확인

- 허깅페이스 파이프라인 객체는 model 속성에 모델 객체를 저장하고, tokenizer 속성에 토큰나이저 객체를 저장
- kobart 모델의 어휘 사전 크기를 확인

```
print(kobart.tokenizer.vocab_size) → 30000
```

- 토큰나이저 객체에 len ( ) 함수를 적용하여 어휘 사전 크기를 확인

```
len(kobart.tokenizer) → 30000
```

- vocab 속성을 통해 전체 어휘 사전 가져오기

```
vocab = kobart.tokenizer.vocab  
len(vocab) → 30000
```

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(31)

### ○ BPE

- vocab 변수는 단순한 파이썬 딕셔너리
  - items ( ) 메서드의 출력을 리스트로 바꾼 다음 맨 처음 토큰 몇 개를 확인
    - 튜플의 리스트가 출력 - 각 튜플은 토큰과 토큰 아이디의 쌍으로 이루어져 있음
    - \_ 문자는 공백을 의미 - 즉 해당 토큰 앞에 공백이 있으므로 단어의 시작 부분을 나타냄

```
list(vocab.items())[:10]
```

→

```
[('_선고', 16904),  
 ('_꾸며', 24443),  
 ('_않았을', 23083),  
 ('處', 7219),  
 ('_계획이라고', 26348),  
 ('_만만', 22616),  
 ('_아저', 22677),  
 ('답', 9892),  
 ('첼', 12818),  
 ('커', 12922)]
```

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(32)

### ○ BPE

- `tokenize( )` 메서드로 샘플 텍스트를 직접 토큰으로 나누기
  - '혼자 만들면서 공부하는 딥러닝'이 8개 토큰으로 나뉘어짐

```
tokens = kobart.tokenizer.tokenize('혼자 만들면서 공부하는 딥러닝')  
print(tokens)
```

→

['\_혼자', '\_만들', '면서', '\_공부하는', '\_', '딥', '러', '닝']

- `convert_tokens_to_ids( )` 메서드를 사용하여 각 토큰에 해당하는 토큰 아이디 찾기

```
kobart.tokenizer.convert_tokens_to_ids(tokens)
```

→

[16814, 14397, 14125, 25429, 1700, 10021, 10277, 9747]

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(33)

### ○ BPE

- `encode( )` 메서드는 문자열에서 토큰 아이디 리스트를 바로 만들어 줌
  - `encode( )` 메서드의 출력을 보면 맨 앞과 뒤에 토큰 아이디 0과 1이 추가됨
    - 이 두 토큰은 문자열의 시작과 끝을 나타냄

```
token_ids = kobart.tokenizer.encode('혼자 만들면서 공부하는 딥러닝')  
print(token_ids)
```

→

[0, 16814, 14397, 14125, 25429, 1700, 10021, 10277, 9747, 1]

- `convert_ids_to_tokens( )` 메서드 - 토큰 아이디를 토큰으로 변환

```
tokens = kobart.tokenizer.convert_ids_to_tokens(token_ids)  
print(tokens)
```

→

['<s>', '\_혼자', '\_만들', '\_면서', '\_공부하는', '\_', '딥', '러', '닝', '</s>']

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(34)

### ○ BPE

- `decode( )` 메서드 - 토큰 리스트를 원래 문자열로 복원

```
kobart.tokenizer.decode(token_ids)
```

→

'<s> 혼자 만들면서 공부하는 딥러닝</s>'



## SECTION 10-2 트랜스포머로 상품 설명 요약하기(35)

### ○ 워드피스

- 워드피스 토큰화는 BPE 토큰화와 매우 비슷함
  - 먼저 훈련 데이터셋에 있는 모든 문자가 포함된 어휘 사전으로 시작해서 가장 많이 등장하는 부분단어 쌍을 어휘 사전에 추가
    - BPE는 단순히 가장 많이 등장하는 부분단어를 선택하지만 워드피스는 부분단어를 구성하는 개별 토큰의 빈도도 고려
  - 예) “ab” 토큰과 “cd” 토큰의 빈도가 각각 10과 200이라면
    - BPE 알고리즘은 “cd” 토큰을 먼저 어휘 사전에 추가
    - 워드피스는 “ab” 토큰의 빈도를 “a”와 “b” 토큰의 빈도로 나누고, 마찬가지로 “cd” 토큰의 빈도를 “c”와 “d” 토큰의 빈도로 나누어 둘 중 큰 값을 가진 토큰을 어휘사전에 추가
  - 이런 방식 때문에 BPE 알고리즘으로 만든 어휘 사전과 조금 다른 어휘 사전을 구성
- 워드피스를 사용하는 대표적인 LLM은 인코더 기반 모델인 BERT

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(36)

### ○ 유니그램

- 유니그램 토큰화에서는 초기에 매우 큰 어휘 사전을 만든 다음 사전에 지정한 어휘 사전 크기에 도달할 때까지 점진적으로 토큰을 제거
  - 초기 어휘 사전은 공백으로 나뉘어진 단어를 부분단어로 쪼개어 추가하거나 BPE 알고리즘을 적용
  - 그다음 어휘 사전에 있는 모든 토큰이 독립적이라 가정하고, (그래서 유니그램) 전체 손실을 가장 적게 증가시키는 토큰을 하나씩 삭제
    - 여기서 손실은 각 토큰의 등장 확률을 곱한 것에 음수를 취한 값
    - 작은 실숫값인 확률을 곱하는 대신 계산하기 쉬운 덧셈으로 바꾸기 위해 로그를 씌움  
즉, 손실은 음의 로그 확률 또는 음의 로그 가능도(negative log likelihood)

$$\begin{array}{c} \text{n개의 확률} \\ \underbrace{\phantom{P_1 \times P_2 \times \dots \times P_n}} \\ - (P_1 \times P_2 \times \dots \times P_n) \rightarrow -\log(P_1 \times P_2 \times \dots \times P_n) \rightarrow -(\log P_1 + \log P_2 + \dots + \log P_n) \end{array}$$

- 일반적으로 유니그램 토큰화는 단독으로 쓰이지 않으며 센토크스피스와 함께 사용

## SECTION 10-2 트랜스포머로 상품 설명 요약하기(37)

### ○ 센텐스피스

- 지금까지 언급한 모든 토큰화 방법은 사전 토큰화(pre-tokenization)
  - 먼저 텍스트를 공백 등을 기준으로 단어로 나눔
  - 하지만 중국어와 같은 일부 언어의 경우 이런 방식이 통하지 않음
- 센텐스피스는 사전 토큰화를 사용하지 않는 방법으로 최신 언어 모델에서 널리 사용
  - 원시 입력 텍스트를 그대로 사용하며 공백을 문자의 하나로 간주
- 센텐스피스는 알고리즘이자 라이브러리이며 실제 어휘 사전 구성은 BPE나 유니그램 알고리즘을 사용

## SECTION 10-2 마무리(1)

### ○ 키워드로 끝나는 핵심 포인트

- 전이 학습
  - 한 데이터셋에서 훈련한 모델을 다른 작업 혹은 다른 데이터셋에 적용하는 방법
- BART
  - 메타에서 공개한 트랜스포머 기반 인코더-디코더 모델
  - 한국어 데이터셋에서 훈련된 KoBART 모델
- 허깅페이스
  - 트랜스포머 모델을 쉽게 사용하고 훈련하기 위한 transformers 패키지를 만든 회사
  - 허깅페이스 사이트에는 다양한 트랜스포머 기반 자연어 처리 모델은 물론 비전과 오디오 작업을 위한 모델과 데이터셋을 무료로 제공
- 토큰화
  - 대규모 언어 모델에 입력하기 위해 텍스트를 더 작은 단어로 쪼개는 과정

## SECTION 10-2 마무리(2)

### ○ 핵심 패키지와 함수

#### - transformers

- pipeline( ) - 허깅페이스 transformers 라이브러리로 간편하게 모델 추론을 할 수 있는 파이프라인 객체를 만들
  - task 매개변수 - 파이프라인 객체로 수행할 작업을 지정 - 'text-classification', 'summarization' 등
  - model 매개변수 - 작업 수행에 사용할 모델을 지정
  - device 매개변수 - 추론에 사용할 하드웨어 장치를 지정
- 파이프라인 객체의 model 속성
  - pipeline( ) 함수를 호출할 때 로드한 모델 객체를 담고 있음
  - 이 모델 객체의 config 속성은 인코더와 디코더 개수, 임베딩 크기 등 LLM 모델을 위한 다양한 설정을 포함
- 파이프라인 객체의 tokenizer 속성
  - pipeline( ) 함수를 호출할 때 로드한 토큰라이저 객체를 담고 있음

## SECTION 10-2 마무리(3)

### ○ 핵심 패키지와 함수

#### - 허깅페이스 토큰라이저 객체의 속성과 메서드

- vocab\_size 속성 - 토큰라이저의 어휘 사전 크기
- vocab 속성 - 토큰라이저의 어휘 사전을 반환
  - 토큰라이저의 get\_vocab( ) 메서드를 호출하는 것과 동일
- tokenize( ) 메서드
  - 문자열을 토큰으로 분할하여 토큰 리스트를 반환
- convert\_tokens\_to\_ids ( ) 메서드
  - 토큰(또는 토큰의 리스트)을 토큰 아이디(또는 토큰 아이디의 리스트)로 변환
- convert\_ids\_to\_tokens ( ) 메서드
  - 토큰 아이디(또는 토큰 아이디의 리스트)를 토큰(또는 토큰 리스트)으로 변환
- encode( ) 메서드
  - 문자열(또는 문자열 리스트)를 토큰으로 분할하여 토큰 아이디의 리스트를 반환
- decode( ) 메서드
  - 토큰 아이디(또는 토큰 아이디의 리스트)를 문자열로 복원

## SECTION 10-2 확인 문제(1)

1 다음 중 오픈소스 LLM 모델이 아닌 것은?

- ① GPT-3
- ② Llama
- ③ Gemma
- ④ Qwen

2 다음 중 인코더-디코더 트랜스포머 모델의 설명으로 옳바른 것은?

- ① 인코더의 개수와 디코더의 개수는 같아야 함
- ② 각각의 인코더 블록에서 나온 출력이 합쳐져서 디코더로 전달됨
- ③ 인코더와 디코더는 각각 독립적인 토큰 임베딩 층을 가지고 있음
- ④ 마지막 인코더 블록의 출력이 모든 디코더 블록에 전달됨

## SECTION 10-2 확인 문제(2)

3 다음 중 토크나이저와 토큰화에 대한 설명으로 옳바르지 않은 것은?

- ① 토크나이저는 토큰과 토큰 아이디를 매핑한 어휘 사전을 가지고 있음
- ② 토크나이저는 토큰을 단어 임베딩으로 변환하는 역할을 수행
- ③ LLM에서 널리 사용하는 토큰화 방법에는 BPE, 워드피스, 유니그램 등이 있음
- ④ 센텐스피스 토큰화를 수행하기 전에 원시 텍스트를 단어로 분할할 필요가 없음



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(1)

## ○ 디코더 기반의 대규모 언어

### - 대표적인 오픈 소스 모델

- 메타 Llama : <https://www.llama.com/>
- 구글 Gemma : <https://ai.google.dev/gemma>
- 마이크로소프트 Phi : <https://azure.microsoft.com/en-us/products/phi/>
- 알리바바 Qwen : <https://qwenlm.github.io/>

### - 대표적인 클로즈드 소스 모델

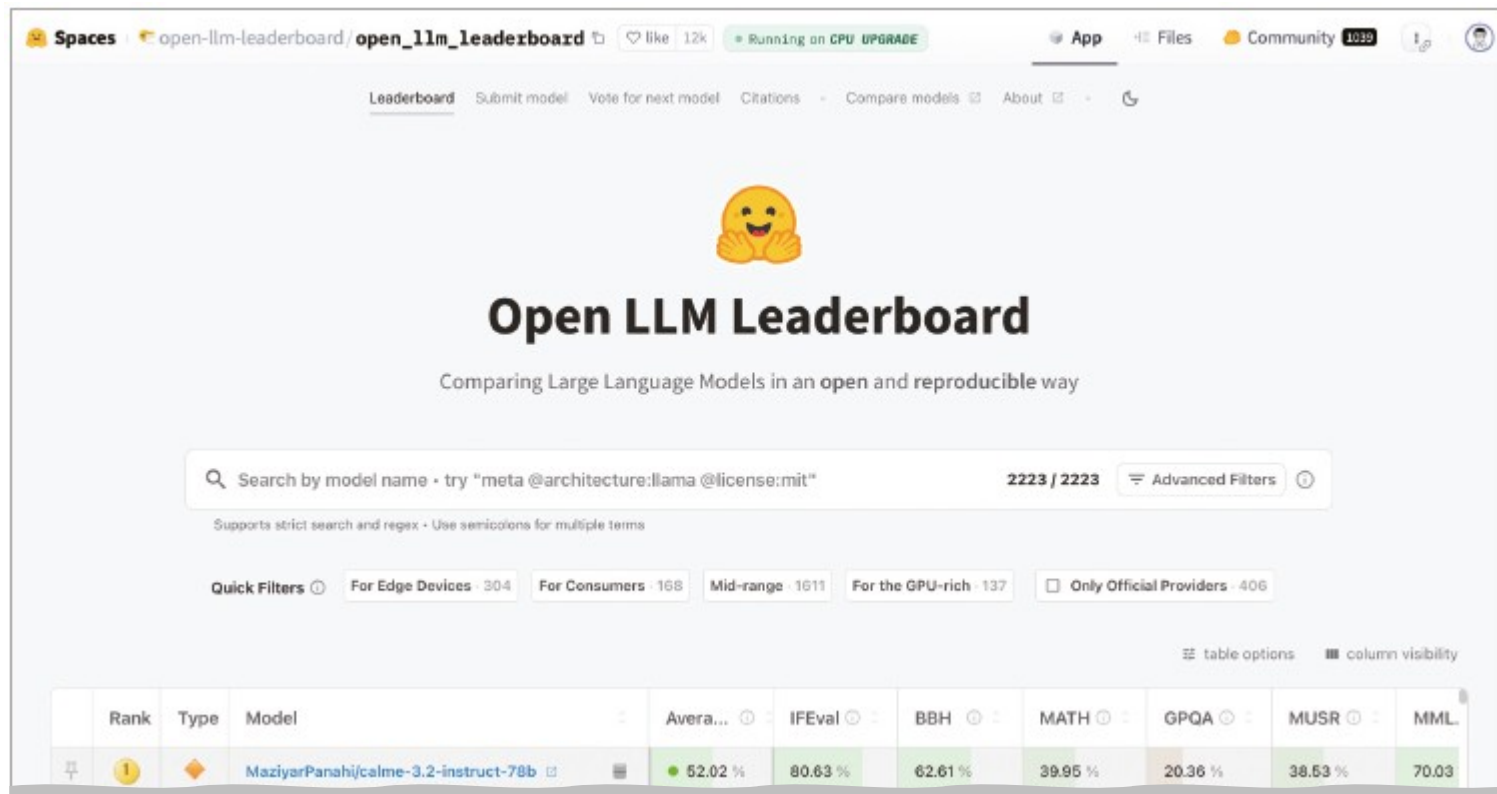
- 오픈AI GPT-4 : <https://chat.com>
- 앤트로픽(Anthropic) Claude : <https://claude.ai/>
- 구글 Gemini : <https://gemini.google.com/>

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(2)

## ○ LLM 리더보드

### - 대규모 언어 모델의 성능을 비교하는 서비스

- 오픈 LLM 리더보드(Open LLM Leaderboard)
  - 허깅페이스에서 제공하는 서비스로, 허깅페이스에 등록된 오픈 소스 LLM의 성능을 비교
- LMSYS 챗봇 아레나 리더보드(LMSYS Chatbot Arena Leaderboard)



The screenshot shows the Open LLM Leaderboard interface. At the top, there's a navigation bar with 'Spaces', 'open-llm-leaderboard/open\_llm\_leaderboard', and a 'Running on CPU UPGRADE' badge. Below the navigation bar, there's a search bar with the text 'Search by model name • try "meta @architecture:llama @license:mit"'. The main title 'Open LLM Leaderboard' is prominently displayed, followed by the subtitle 'Comparing Large Language Models in an open and reproducible way'. Below the title, there are 'Quick Filters' for different device types: 'For Edge Devices: 304', 'For Consumers: 168', 'Mid-range: 1611', 'For the GPU-rich: 137', and 'Only Official Providers: 406'. At the bottom, a table lists the top models with their performance metrics across various benchmarks.







Rank	Type	Model	Avera...	IFEval	BBH	MATH	GPQA	MUSR	MML
1	🔥	MaziyerPanahi/calme-3.2-instruct-78b	52.02 %	80.63 %	62.61 %	39.95 %	20.36 %	38.53 %	70.03

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(3)

## ○ LLM 리더보드

### - 오픈 LLM 리더보드(Open LLM Leaderboard)

- 모델의 순위의 기준의 이해를 위해 리더보드의 구성 요소를 확인
- 목록의 왼쪽의 Type 옆에 나타난 아이콘 - 모델의 종류

	대규모 데이터셋에서 사전 훈련된 베이스 모델(또는 파운데이션 모델 foundation model )
	특정 데이터셋에서 미세 튜닝된 모델
	대화 기능을 위해 미세 튜닝된 모델
	여러 모델을 병합하여 만든 모델
	텍스트, 이미지, 오디오 등과 같이 여러 종류의 데이터를 다룰 수 있는 모델
	다양한 데이터셋을 사용해 추가로 훈련한 베이스 모델

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(4)

## ○ LLM 리더보드

### - 오픈 LLM 리더보드(Open LLM Leaderboard)

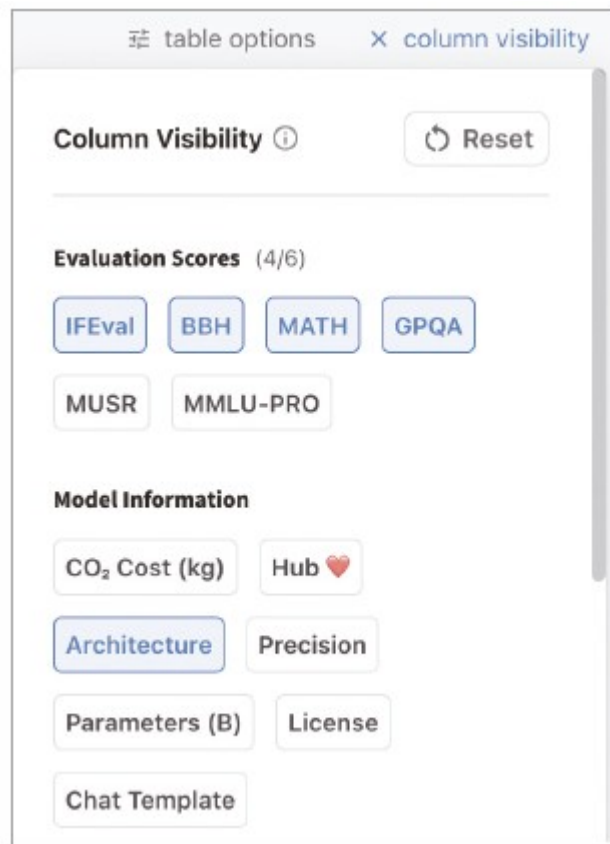
- Average 얻은 모든 벤치마크에서 얻은 점수를 평균한 값
  - IFEval(Instruction-Following Evaluation): 약 500개의 프롬프트10를 선택하여 언어 모델이 프롬프트의 지시를 얼마나 잘 따르는지 평가한 값
  - BBH(Big Bench Hard): 빅 벤치(Big-Bench) 평가의 하위 집합으로 다단계 추론 능력을 평가하는 어려운 과제로 구성
  - MATH(Mathematics Aptitude Test of Heuristics): 수학 문제 해결 능력을 평가하는 벤치마크
    - 12,500개의 문제로 구성되어 있고 대수학, 정수론, 조합, 확률, 미적분 등의 문제를 풀어야 함
  - GPQA(Graduate-Level Google-Proof Q&A): 화학, 생물학, 물리학 분야에서 박사 수준의 448개의 객관식 문제를 푸는 벤치마크
  - MUSR(Multistep Soft Reasoning): 자연어로 묘사된 추론 문제를 푸는 벤치마크
    - 예) 1,000단어 길이의 미스터리 문제를 풀어야 하는 과제
  - MMLU-Pro(Massive Multitask Language Understanding - Professional): 대규모 언어 모델의 언어 이해와 추론 능력을 평가하기 위한 벤치마크로, 기존의 MMLU보다 더 복잡하고 어려운 12,000개의 문제로 구성
  - CO<sub>2</sub> Cost: 모델을 평가하는 데 사용된 전력을 생산하기 위해 배출된 탄소의 양(Kg)

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(5)

## ○ LLM 리더보드

### - 오픈 LLM 리더보드(Open LLM Leaderboard)

- 리더보드는 기본적으로 Average 열의 값을 기준으로 정렬
  - 다른 열을 기준으로 정렬하려면 열 이름 옆에 있는 아이콘을 클릭하여 내림차순이나 오름차순으로 표시
  - 테이블 오른쪽 위의 아이콘 - 표시하고 싶은 열을 추가하거나 제외



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(6)

## ○ LLM 리더보드

### - 오픈 LLM 리더보드(Open LLM Leaderboard)

- 오른쪽 위 아이콘을 클릭한 다음 Architecture 버튼을 클릭하면
  - 목록에 있는 모델이 사용한 파운데이션 모델 또는 베이스 모델을 Architecture 열에 보여줌

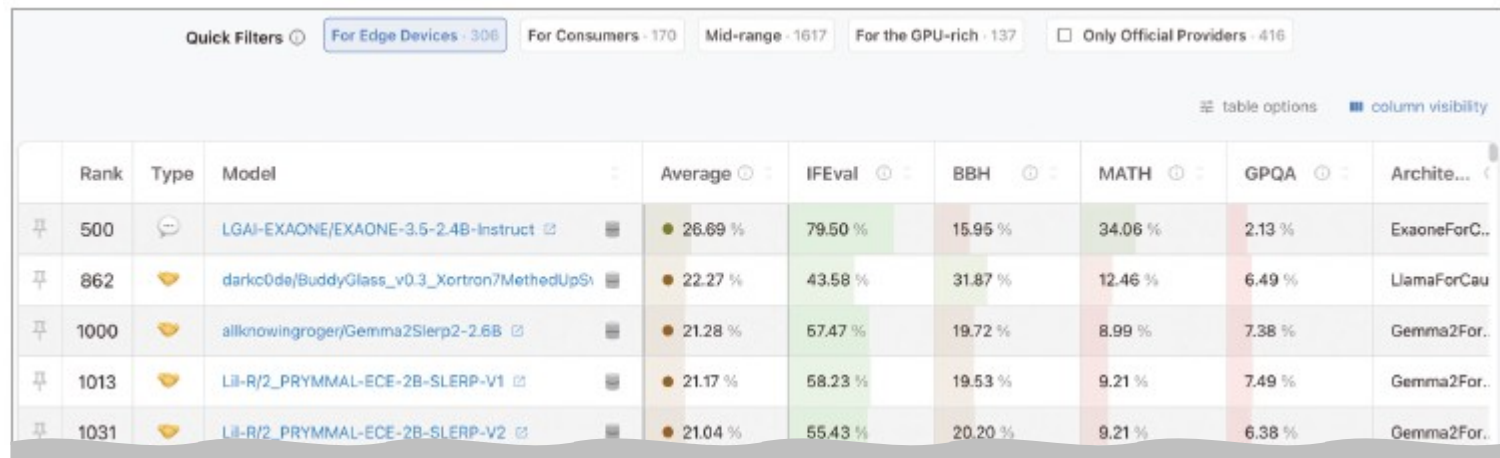
	Rank	Type	Model	Average ①	IFEval ①	BBH ①	MATH ①	GPQA ①	Archite...
푸	1	◆	MaziyarPanahi/calme-3.2-instruct-78b	● 52.02 %	80.63 %	62.61 %	39.95 %	20.36 %	Qwen2ForCa
푸	2	💬	dfurman/CalmeRys-78B-Orpo-v0.1	● 51.24 %	81.63 %	61.92 %	40.71 %	20.02 %	Qwen2ForCa
푸	3	💬	MaziyarPanahi/calme-3.1-instruct-78b	● 51.20 %	81.36 %	62.41 %	38.75 %	19.46 %	Qwen2ForCa
푸	4	💬	MaziyarPanahi/calme-2.4-rys-78b	● 50.71 %	80.11 %	62.16 %	40.41 %	20.36 %	Qwen2ForCa
푸	5	◆	rombodawg/Rombos-LLM-V2.5-Qwen-72b	● 45.91 %	71.55 %	61.27 %	50.68 %	19.80 %	Qwen2ForCa
푸	6	◆	zetasepic/Qwen2.5-72B-Instruct-abliterated	● 45.29 %	71.53 %	59.91 %	46.15 %	20.92 %	Qwen2ForCa
푸	7	◆	dnhkng/RYS-XLarge	● 45.13 %	79.96 %	58.77 %	41.24 %	17.90 %	Qwen2ForCa
푸	8	◆	rombodawg/Rombos-LLM-V2.5-Qwen-32b	● 44.57 %	68.27 %	58.26 %	41.99 %	19.57 %	Qwen2ForCa
푸	9	💬	MaziyarPanahi/calme-2.1-rys-78b	● 44.56 %	81.36 %	59.47 %	38.90 %	19.24 %	Qwen2ForCa

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(7)

## ○ LLM 리더보드

### - 오픈 LLM 리더보드(Open LLM Leaderboard)

- 학습에서는 코랩에서 예제를 실행해야 하므로, 파라미터 개수가 작은 모델을 사용하는 것이 좋음
- 리더보드 중앙 검색 창 아래 'Quick Filters' 버튼
  - 모델 파라미터 크기에 따라 순서대로 네 개의 버튼이 제공
    - For Edge Devices : 30억 개 이하
    - For Consumers : 30억 개 ~ 70억 개
    - Mid-range : 70억 개 ~ 650억 개
    - For the GPU-rich : 650억 개 이상
- 이 중 'For Edge Devices'를 선택



Quick Filters									
For Edge Devices - 306									
For Consumers - 170									
Mid-range - 1617									
For the GPU-rich - 137									
<input type="checkbox"/> Only Official Providers - 416									
table options column visibility									
	Rank	Type	Model	Average	IFEval	BBH	MATH	GPQA	Archite...
푸	500		LGAI-EXAONE/EXAONE-3.5-2.4B-Instruct	26.69 %	79.50 %	15.95 %	34.06 %	2.13 %	ExaoneForC..
푸	862		dark0de/BuddyGlass_v0.3_Xortron7MethodUpS	22.27 %	43.58 %	31.87 %	12.46 %	6.49 %	LlamaForCau
푸	1000		allknowingroger/Gemma2Slerp2-2.6B	21.28 %	57.47 %	19.72 %	8.99 %	7.38 %	Gemma2For..
푸	1013		Li-R/2_PRYMMAL-ECE-2B-SLERP-V1	21.17 %	58.23 %	19.53 %	9.21 %	7.49 %	Gemma2For..
푸	1031		Li-R/2_PRYMMAL-ECE-2B-SLERP-V2	21.04 %	55.43 %	20.20 %	9.21 %	6.38 %	Gemma2For..

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(8)

### ○ EXAONE의 특징

- EXAONE은 국내에서 오픈 소스로 공개된 파운데이션 모델
  - 한국어와 영어를 잘 이해하며 다양한 작업을 수행할 수 있는 모델
  - 모델의 3.5 버전은 최신 LLM에서 채택하는 여러 기술을 사용
- EXAONE은 디코더 기반 트랜스포머 모델
  - BART처럼 인코더의 출력을 전달받는 크로스 어텐션 층이 없다는 점을 유념
- EXAONE은 최신 LLM에서 널리 사용하는 그룹 쿼리 어텐션(grouped query attention)을 사용
  - 멀티 헤드 어텐션의 변형



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(9)

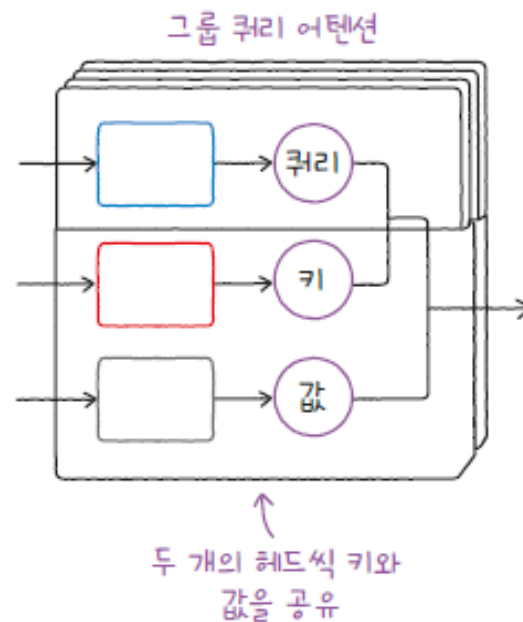
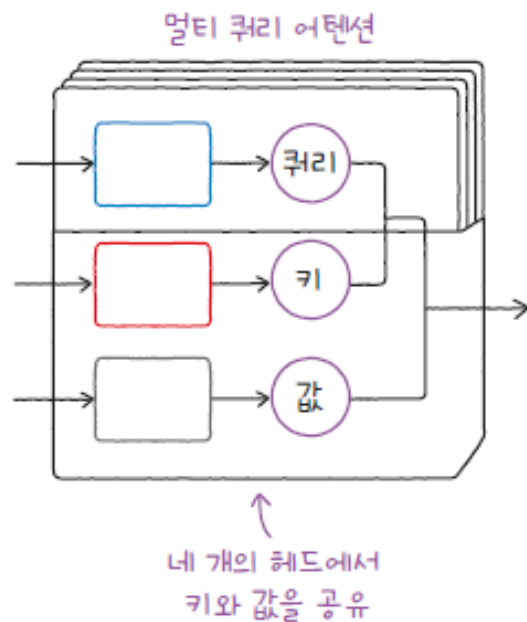
## ○ EXAONE의 특징

- 멀티 쿼리 어텐션(multi-query attention)
  - 멀티 헤드 어텐션에서 키와 값을 모든 헤드에서 공유하는 방식
  - 디코더는 하나의 토큰을 생성한 후, 그 토큰을 입력의 끝에 이어 붙인 다음 다시 모델에 입력해 다음 토큰을 생성 - 자기회귀 모델
    - 이 방식에서는 디코더가 하나의 토큰을 생성할 때마다 이전에 처리했던 토큰들을 매번 다시 계산해야 하므로, 언뜻 보면 계산 낭비처럼 보임
    - 이 문제를 해결하기 위해 어텐션 층에서 키와 값을 캐시에 저장하고 다음 토큰을 생성할 때 재사용하는 기법이 등장
    - 그러나 트랜스포머에 입력할 수 있는 최대 입력 길이를 늘리려면 캐시의 크기도 자연스럽게 커질 수밖에 없음
  - 문제 해결 - 멀티 헤드 어텐션에서 키와 값을 모든 헤드에서 공유하는 방식이 등장
- 그룹 쿼리 어텐션
  - 모든 헤드에서 키와 값을 공유하지 않고, 몇 개의 헤드씩 나눠서 공유하는 방식

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(10)

### ○ EXAONE의 특징

- 멀티 쿼리 어텐션과 그룹 쿼리 어텐션은 키와 값을 만드는 밀집층의 개수가 줄어들기 때문에 전체 모델의 파라미터 개수를 줄이는 효과
- 상대적으로 크기가 작은 LLM에서 널리 사용
- EXAONE은 24억, 78억, 320억 파라미터 버전에서 모두 그룹 쿼리 어텐션을 사용



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(11)

### ○ EXAONE의 특징

#### - 실루(SiLU) 활성화 함수

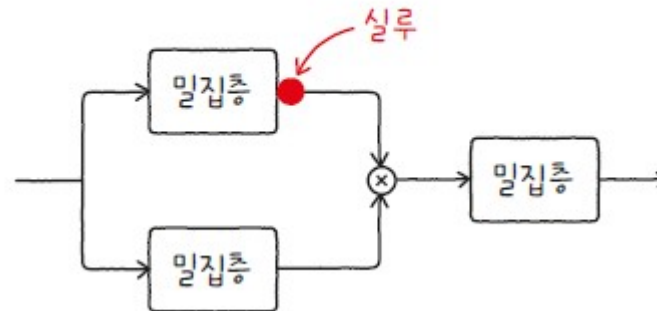
- 어텐션 층 다음에 등장하는 피드포워드 네트워크에서는 최근 LLM에서 많이 사용되는 사용
- 함수는 밀집층의 출력에 시그모이드 함수를 적용한 다음 이 결과에 원래 출력을 다시 곱함
- 실루 함수는 렐루와 비슷한 형태를 가지며, 젤루와 마찬가지로 원점에서도 미분이 가능

시그모이드 함수

$$\text{SiLU}(x) = x \cdot \sigma(x) = x \left( \frac{1}{1 + e^{-x}} \right)$$

- 일반적으로 실루 함수를 적용할 때 피드포워드 네트워크의 첫 번째 밀집층을 두 개로 나누어 하나는 실루 함수를 적용하고, 다른 하나는 활성화 함수를 적용하지 않음 - 그 후, 이 두 출력을 곱함

피드포워드 네트워크



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(12)

## ○ EXAONE의 특징

- RMS 정규화(root mean square normalization)
  - 층 정규화의 변종 - 정규화를 할 때 평균을 구하지 않는 방법
  - 2장에서 표준점수를 계산할 때 - 입력에서 평균을 빼고 표준편차로 나눔
    - 층 정규화도 기본적으로 이와 같은 방식을 사용
  - RMS 정규화는 입력에서 평균을 빼지 않고, 표준편차를 구할 때도 평균을 사용하지 않는 방법

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(13)

## ○ EXAONE의 특징

### - 정규화의 예

- 다음과 같이 다섯 개의 원소를 가진 배열을 정규화한다고 가정
  - 1에서 5까지의 값으로 이루어져 있으므로 평균은 3
  - 분산은 각 원소에서 평균을 뺀 후 제곱한 후, 전체 원소 개수로 나누어 계산
  - 분산을 제곱근하면 표준편차
  - 정규화를 할 때는 각 원소에서 평균을 빼고 표준편차로 나눔

1	2	3	4	5
---	---	---	---	---

$$\text{분산} = \frac{1}{5} \left( (1-3)^2 + (2-3)^2 + (3-3)^2 + (4-3)^2 + (5-3)^2 \right) = 2$$

$$\text{표준편차} = \sqrt{2} = 1.41$$

$$\text{첫 번째 원소의 정규화} : \frac{1-3}{1.41} = -1.42$$

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(14)

## ○ EXAONE의 특징

### - RMS 정규화의 예

- 앞의 과정에서 평균을 사용하지 않는 방법

- 분산은 각원소의 제곱을 모두 더한 다음 전체 원소 개수로 나눔
- 정규화를 할 때는 각 원소에서 평균을 빼지 않고 원소에 그대로 표준편차를 나누어 줌

1	2	3	4	5
---	---	---	---	---

$$\text{분산} = \frac{1}{5} (1^2 + 2^2 + 3^2 + 4^2 + 5^2) = 11$$

$$\text{표준편차} = \sqrt{11} = 3.32$$

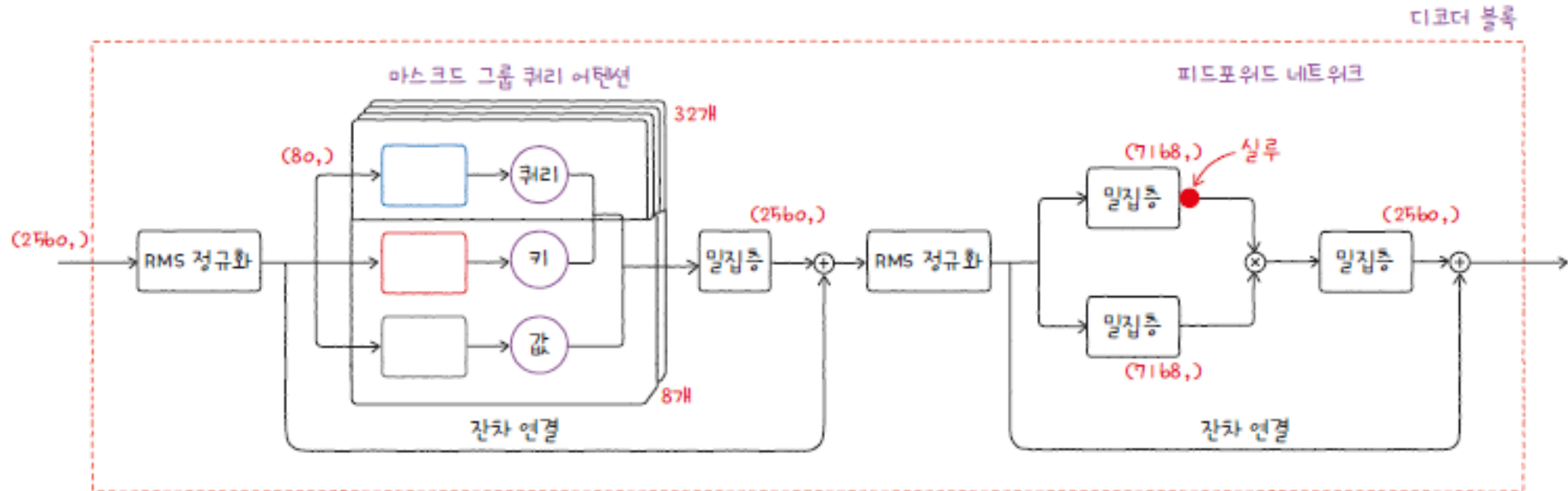
$$\text{첫 번째 원소의 정규화} : \frac{1}{3.32} = 0.3$$

- RMS 정규화를 사용하면 평균을 계산하지 않아도 되므로 계산 속도가 빠름
- 또 평균을 사용해 데이터를 중심에 맞추지 않아도 모델의 성능에 큰 영향을 미치지 않는다고 알려짐

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(15)

### ○ EXAONE의 특징

- 최근 LLM은 이런 정규화를 어텐션 층 다음이 아니라 어텐션 이전에 두는 경향
- EXAONE도 마찬가지로 어텐션 층 이전과 피드포워드 네트워크 이전에 RMS 정규화를 적용
  - 24억 파라미터 버전을 기준으로 이런 요소를 디코더 블록에 나타내면 아래 그림과 같음



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(16)

## ○ EXAONE의 특징

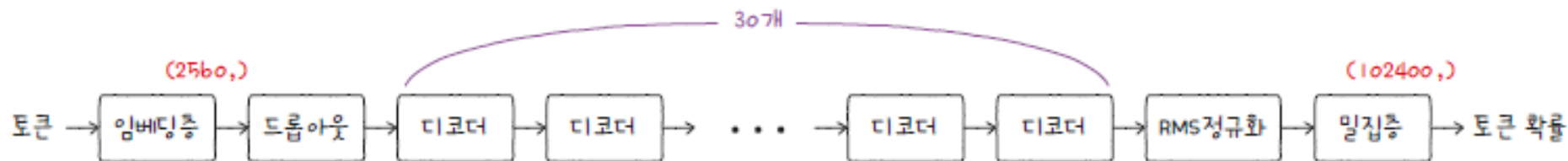
- 디코더 블록은 층 정규화부터 시작
  - 24억 파라미터 버전의 은닉 벡터 크기는 2,560 - 이를 80개씩 나누어 32개의 헤드에 입력
  - 그룹 쿼리 어텐션을 사용하며 키와 값의 헤드는 8개
  - 어텐션 층이 출력한 벡터의 크기는 다시 2,560이 되고 잔차 연결을 지나 다시 층 정규화를 거침
- 피드포워드 네트워크
  - 실루 활성화 함수를 사용하며 두 개의 밀집층 중에서 하나에만 적용
  - 두 밀집층은 은닉 벡터의 크기를 늘려 7,168로 만듦
  - 두 밀집층의 결과를 곱한 후 마지막 밀집층에서 은닉 벡터의 크기는 다시 2,560으로 줄어듦
  - 이런 디코더 블록을 30개 쌓음



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(17)

## ○ EXAONE의 특징

### - 24억 파라미터 버전의 EXAONE 전체 구조



- 토큰 아이디가 임베딩 층을 통과해 2,560차원의 벡터로 변환되고 드롭아웃 층을 지남
- 그다음 앞서 소개한 디코더 블록 30개를 통과
- 마지막에 층 정규화를 거치고 밀집층을 통과하면서 어휘 사전 크기인 102,400 크기의 벡터를 출력
- 이 출력에 소프트맥스 함수를 적용하면 어휘 사전에 있는 모든 토큰에 대한 확률처럼 생각할 수 있음

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(18)

## ○ EXAONE의 특징

- EXAONE은 별도의 밀집층을 사용
  - 위치 임베딩이 없음
  - EXAONE이 사용하는 위치 임베딩은 디코더 블록의 어텐션 층 안에 포함
- 로터리 위치 임베딩(rotary position embedding, RoPE)
  - EXAONE을 비롯해 최근 LLM에서 널리 사용되는 위치 임베딩 방식
  - 로터리 위치 임베딩은 쿼리와 키 벡터를 서로 다른 각도로 회전
    - 이렇게 회전시킨 두 벡터를 곱하면 그 결과에 두 벡터의 상대적인 각도 차이를 인코딩
- 절대 위치 인코딩
  - 기존의 위치 인코딩과 위치 임베딩 - 토큰의 절대적인 위치 인덱스를 벡터로 변
- 상대 위치 인코딩
  - 로터리 위치 임베딩은 쿼리와 키의 상대적인 각도 차이를 표현
  - 로터리 위치 임베딩은 별도의 위치 임베딩 벡터를 만들지 않기 때문에 계산이 간단하고 트랜스포머 모델의 성능도 향상시킨다고 알려짐

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(19)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

#### - EXAONE-3.5의 24억 파라미터 버전을 사용

- EXAONE-3.5의 전체 모델은 허깅페이스 사이트(<https://bit.ly/4gsiHHF>)를 참고
- AutoTokenizer 클래스의 from\_pretrained ( ) 클래스 메서드로 EXAONE의 토큰라이저를 로드
  - 불러올 토큰라이저 이름은 모델 이름을 지정할 때와 동일하게 허깅페이스의 경로를 전달

```
from transformers import AutoTokenizer  
  
exaone_tokenizer = AutoTokenizer.from_pretrained(  
    "LGAI-EXAONE/EXAONE-3.5-2.4B-Instruct")
```

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(20)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

- pipeline ( ) 함수의 tokenizer 매개변수로 exaone\_tokenizer를 전달

```
from transformers import pipeline

pipe = pipeline(task="text-generation",
                model="LGAI-EXAONE/EXAONE-3.5-2.4B-Instruct",
                tokenizer=exaone_tokenizer,
                device=0, trust_remote_code=True)
```

- tokenizer 외에도 trust\_remote\_code 매개변수가 추가됨
- 허깅페이스에 있는 모델은 transformers 패키지의 코드 외에 독자적인 코드로 모델을 정의할 수 있음
  - EXAONE이 바로 이런 경우에 해당
- 이런 코드는 실행하기 전에 사용자에게 실행 여부를 묻음
  - trust\_remote\_code를 True로 설정하면 코드를 신뢰한다고 가정하고 일일이 실행할지 여부를 묻지 않음

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(21)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

#### - 채팅 템플릿 만들기

- 채팅 템플릿은 딕셔너리의 리스트로 구성
- 딕셔너리의 키
  - “role” - “system”과 “user”와 같은 대화 상대의 역할을 지정
  - “content” - 실제 메시지 내용
- 예) 다음과 같이 “role”을 “system”으로 지정하고 EXAONE 모델이 어떤 역할을 맡게 되는 지를 기술
  - 여기에서는 한빛 마켓에 올라온 문의 글에 자동으로 답변하는 역할
  - 그다음 “role”을 “user”로 지정하고 실제 상품 문의와 같은 메시지를 적음
- 이렇게 두 개의 딕셔너리를 만든 다음 리스트로 연결하여 채팅 템플릿 구성을 종료

```
messages = [  
    {"role": "system",  
     "content": "너는 쇼핑몰 홈페이지에 올라온 질문에 대답하는 Q&A 챗봇이야.\  
                확정적인 답변을 하지 말고 제품 담당자가 정확한 답변을 하기 위해 \  
                시간이 필요하다는 간단하고 친절한 답변을 생성해줘."},  
    {"role": "user", "content": "이 다이어리에 내년도 공휴일이 표시되어 있나요?"}  
]
```

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(22)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

#### - 채팅 템플릿 만들기

##### • 파이프라인 객체를 호출

- 호출 방법은 이전 절과 동일하지만 이번에는 너무 긴 텍스트가 생성되지 않도록 max\_new\_tokens를 200으로 지정

```
pipe(messages, max_new_tokens=200)
```



```
[{'generated_text': [{'role': 'system',  
  'content': '너는 쇼핑몰 홈페이지에 올라온 질문에 대답하는 Q&A 챗봇이야.  
확정적인 답변을 하지 말고 제품 담당자가 정확한 답변을 하기 위해 시간이 필요하다는 간단하고 친절한 답  
변을 생성해줘.'},  
  {'role': 'user', 'content': '이 다이어리에 내년도 공휴일이 표시되어 있나요?'},  
  {'role': 'assistant',  
  'content': '안녕하세요! 다이어리에 내년의 공휴일이 미리 표시되어 있는지에 대해 정확한 답변을  
드리기 위해서는 제품 담당자에게 확인이 필요합니다. 현재로서는 직접 확인이 어려우니, 저희가 안내드릴 수  
있는 방법으로는 고객센터에 연락하시거나, 제품 페이지 내의 문의 게시판을 통해 질문해 보시는 것이 좋을  
것 같습니다. 담당자분께서 빠르게 답변해 주실 거예요! 감사합니다.'}]}]
```

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(23)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

- 모델이 생성한 텍스트만 출력하려면 `return_full_text` 매개변수를 `False`로 지정

```
pipe(messages, max_new_tokens=500, return_full_text=False) →
```

[{'generated\_text': '안녕하세요! 다이어리에 내년의 공휴일이 미리 표시되어 있는지에 대해 정확한 답변을 드리기 위해서는 제품 담당자에게 확인이 필요합니다. 현재로서는 직접 확인이 어려우니, 저희가 안내드릴 수 있는 방법으로는 고객센터에 연락하시거나, 제품 페이지 내의 문의 게시판을 통해 질문해 보시는 것이 좋을 것 같습니다. 담당자분께서 빠르게 답변해 주실 거예요! 감사합니다.'}]

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(24)

### ○ EXAONE-3.5로 상품 질문에 대한 대답 생성하기

#### - 앞의 두 결과를 보면 출력 내용이 동일

- 이는 다음 토큰을 선택할 때 무조건 가장 높은 확률의 토큰을 선택하기 때문임
- 조금 확률적으로 토큰을 선택하고 싶다면 `do_sample` 매개변수를 `True`로 지정  
그리고 반환된 결과에서 'generated\_text' 키만 출력

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True)  
print(output[0]['generated_text'])
```



안녕하세요! 다이어리에 내년의 모든 공휴일이 미리 표시되어 있는지 확인해 드리기 위해서는 제품 담당자님의 직접적인 확인이 가장 정확할 거예요. 하지만 현재로서는 저희가 실시간으로 업데이트된 정보를 제공하는 데 한계가 있어요. 시간이 좀 걸리더라도 곧 연락드리거나 담당자님께서 알려주시면 도와드리겠습니다. 궁금한 점이 더 있으시다면 알려주세요!

`do_sample` 매개변수를 `True`로 지정하니 답변이 조금 더 자연스럽게 생성



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(25)

### + 여기서 잠깐

- do\_sample 매개변수를 사용하면 코드를 실행할 때마다 다른 토큰이 선택될 수 있음
  - 책에 나온 출력 결과가 깃허브의 노트북과 다를 수 있음
- 만약 실행할 때마다 동일한 결과를 얻고 싶다면 다음처럼 set\_seed() 함수를 사용하여 난수 발생의 시드 값을 지정

```
from transformers import set_seed

set_seed(42)
```

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(26)

### ○ 토큰 디코딩 전략

#### - 로짓(logit)

- 디코더가 출력하는 것은 확률이 아니라, 어휘 사전에 들어 있는 각 토큰에 대한 점수
- 4장에서 배운 로지스틱 회귀처럼, 이 점수에 소프트맥스 함수를 적용하면 확률로 변환할 수 있음
- 로짓 - 소프트맥스 함수를 적용하기 전의 값

#### - 디코더가 출력한 로짓 중에서 어떤 토큰을 선택할지는 LLM 모델이 아닌, transformers 같은 딥러닝 프레임워크의 몫

- 프레임워크마다 이 과정은 조금씩 다를 수 있기 때문에 같은 모델, 같은 프롬프트, 같은 디코딩 옵션을 사용하더라도 출력된 텍스트가 다를 수 있음

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(27)

### ○ 토큰 디코딩 전략

- transformers 패키지를 기준으로 디코딩 전략을 학습
  - 다른 패키지에서는 디코딩 순서나 적용 방식이 다를 수 있음
- 가장 간단한 디코딩 방식은 `do_sample` 매개변수가 기본값 `False`일 때
  - 이 경우 가장 높은 확률을 가진 토큰 하나를 선택
  - 그렇기 때문에 프롬프트가 같으면 모델을 여러 번 실행해도 항상 같은 대답을 얻을 수 있음
  - 그리디 서치(greedy search)

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(28)

### ○ 토큰 디코딩 전략

- transformers 패키지를 기준으로 디코딩 전략을 학습
  - 다른 패키지에서는 디코딩 순서나 적용 방식이 다를 수 있음
- 가장 간단한 디코딩 방식은 `do_sample` 매개변수가 기본값 `False`일 때
  - 이 경우 가장 높은 확률을 가진 토큰 하나를 선택
  - 그렇기 때문에 프롬프트가 같으면 모델을 여러 번 실행해도 항상 같은 대답을 얻을 수 있음
  - 그리디 서치(greedy search)

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(29)

### ○ 기본 샘플링

- 디코더가 출력한 로짓 중에서 한 토큰의 로짓이 아주 크다고 가정

- 예) 다음처럼 다섯 개의 토큰에 대한 로짓을 얻었다고 가정

- 마지막 값이 다른 네 개의 값보다 월등히 큼

```
import numpy as np
logits = np.array([1, 2, 3, 4, 100])
```

- 이 배열을 소프트맥스 함수에 통과 - 4장에서 했던 것처럼 사이파이의 `softmax()` 함수를 사용

```
from scipy.special import softmax
```

```
probas = softmax(logits)
print(probas)
```



```
[1.01122149e-43 2.74878501e-43 7.47197234e-43 2.03109266e-42
1.00000000e+00]
```

- 마지막 원소의 확률이 거의 1에 가깝고 나머지 원소의 확률은 0에 가까움
- 이런 확률 분포에서 하나의 토큰을 랜덤하게 선택하면 거의 항상 마지막 원소가 선택

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(30)

## ○ 기본 샘플링

- 디코더가 출력한 로짓 중에서 한 토큰의 로짓이 아주 크다고 가정

- 넘파이의 `random.multinomial( )` 함수를 사용 - 주어진 확률 분포를 바탕으로 샘플링을 실행
- 여기서는 100번을 실행

```
np.random.multinomial(100, probas)
```

→ `array([ 0, 0, 0, 0, 100])`

- 100번을 시도했는데 100번 모두 마지막 원소가 선택
- 처음 네 개의 원소가 선택된 횟수는 모두 0
- 이렇게 되면 그리디 서치를 적용하는 것과 별반 다르지 않음

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(31)

### ○ 기본 샘플링

#### - 소프트맥스 함수가 만드는 확률 분포를 조금 변형

##### • logits을 100으로 나눈 다음 다시 실행

```
probas = softmax(logits/100)  
np.random.multinomial(100, probas)
```

→ array([16, 18, 12, 14, 40])

- 여전히 마지막 원소가 선택되는 경우가 높지만, 이전과 달리 처음 네 개의 원소도 어느 정도 선택됨
- 이렇게 로짓을 1보다 큰 값으로 나누면 확률 분포가 조금 더 부드러워져 다른 원소가 선택될 가능성이 높아지기 때문임
- 반대로 1보다 작은 값으로 나누어 주면 확률 분포가 더 결정적으로 바뀌고 가장 큰 로짓을 선택하는 그리디 서치와 비슷하게 동작하게 됨

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(32)

### ○ 기본 샘플링

- 온도 파라미터 - 선택의 다양성을 증가시키거나 줄이는 역할을 하는 값
  - 파이프라인 객체를 사용할 때 temperature 매개변수로 온도값을 조절할 수 있으며, 기본값은 1
  - 온도를 높였을 때 어떤 텍스트가 생성되는지 확인

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, temperature=10.0)  
print(output[0]['generated_text'])
```



저는 해당Qy 관리자 입장이지않그래서요qing 특정 브랜드 또는 제  
조업의 공휴일 포함사항들 자세한 조회 권한 없으므로 공식 판매처  
인 판매처혹은회사 페이지의 Calendar 섹션까지 직접 검색recommen  
sion 부탁드립니다 그런 디테일 확인해봐 보아지에 확실히 정확히  
될 수 박니를 알 수도요... 거기엔 언제나 전문가, 예 쇼핑몰측이  
나 해당 제작 다이어리 관계자하시분들 의견들이 담겨볼 시간 필요  
하셨겠다 이해 부탁드립니다 감사바겠! 🤗🏠👉🏠! 🏠★.answer.n  
ote#DiaryContent#ChR10NlogicalEntries 정확 내용 제공 위해 담당  
자 요청 대기 부탁드립니다 🤗🏠👉🏠 #NeedAssistantRightApprovalsF  
irst.handle. 🏠 🏠



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(33)

### ○ 기본 샘플링

- 온도 파라미터 - 선택의 다양성을 증가시키거나 줄이는 역할을 하는 값
  - 온도 파라미터를 낮춰서 가장 큰 로짓을 가진 토큰에 높은 가능성을 부여

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, temperature=0.001)  
print(output[0]['generated_text'])
```



안녕하세요! 다이어리에 내년의 공휴일이 미리 표시되어 있는지에 대해 정확한 답변을 드리기 위해서는 제품 담당자에게 확인이 필요합니다. 현재로서는 직접 확인이 어려우니, 저희가 안내드릴 수 있는 방법으로는 고객센터에 연락하시거나, 제품 페이지 내의 문의 게시판을 통해 질문해 보시는 것이 좋을 것 같습니다. 담당자 분께서 빠르게 답변해 주실 거예요! 감사합니다.

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(34)

### ○ top-k 샘플링

- 모델이 출력한 로짓을 기준으로 최상위 k개의 토큰을 선택하는 방법
  - 이후 선택된 토큰의 로짓만 소프트맥스 함수에 통과
  - 가능성이 높은 몇 개의 토큰 중에서 하나를 선택하는 방법
- 파이프라인 객체를 호출할 때 top\_k 매개변수를 지정
  - 이 매개변수에는 1보다 큰 정수를 지정
  - 예) 상위 10개의 토큰을 선택하도록 지정

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, top_k=10)  
print(output[0]['generated_text'])
```



네, 저희 제품 담당자가 확인 후 답변을 드리겠습니다. 공휴일 정보는 연도에 따라 변경될 수 있으니 잠시 후에 다시 확인해 주시거나, 저희 고객센터로 연락주시면 더 신속하게 도움을 드리겠습니다. 감사합니다!

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(35)

### ○ top-k 샘플링

- transformers 패키지는 온도 파라미터를 top-k 샘플링이나 top-p 샘플링보다 먼저 적용
  - 즉, 모델이 생성한 로짓에 대해 temperature를 적용해 분포를 조정한 다음 top\_k 매개변수 값만큼 최상위 k개의 로짓을 선택하고, 마지막으로 소프트맥스 함수가 적용
  - 앞선 코드에서는 temperature 매개변수를 지정하지 않았기 때문에 기본값 1.0이 적용
- temperature 값을 조절하면 모델이 생성하는 텍스트의 다양성을 조정할 수 있음
  - 예) 이전과 동일하게 temperature 매개변수를 10.0으로 높여서 결과를 확인

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, top_k=10, temperature=10.0)  
print(output[0]['generated_text'])
```



죄송한데요 말씀해 주세요! 제 정보력으로만선별로 어떤 공휴일이 적용 될지 정확이알아내기에 한계 있어서  
고객센터에 확인하시라고 제안드린대요~ 직원분이 정확하고 자세한정당하시고 답변 드릴게요! 빠르실 거라 생  
각되지만 좀 기다려줘야 할까 봅니다.

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(36)

### ○ top-p 샘플링

- top-k 샘플링과 다르게 최상위 토큰의 개수를 고정하는 것이 아니라 확률 순으로 토큰을 나열한 후 사전에 지정한 확률만큼만 최상위 토큰을 선택하는 방식
- top-p 샘플링을 뉴클리어스 샘플링(nucleus sampling)이라고도 함



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(37)

### ○ top-p 샘플링

- 상위 90%의 확률까지만 선택한다고 가정
  - top-p 샘플링을 적용하려면 top\_p 매개변수에 0.0보다 크고 1.0보다 작은 실숫값을 지정

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, top_p=0.9)  
print(output[0]['generated_text'])
```



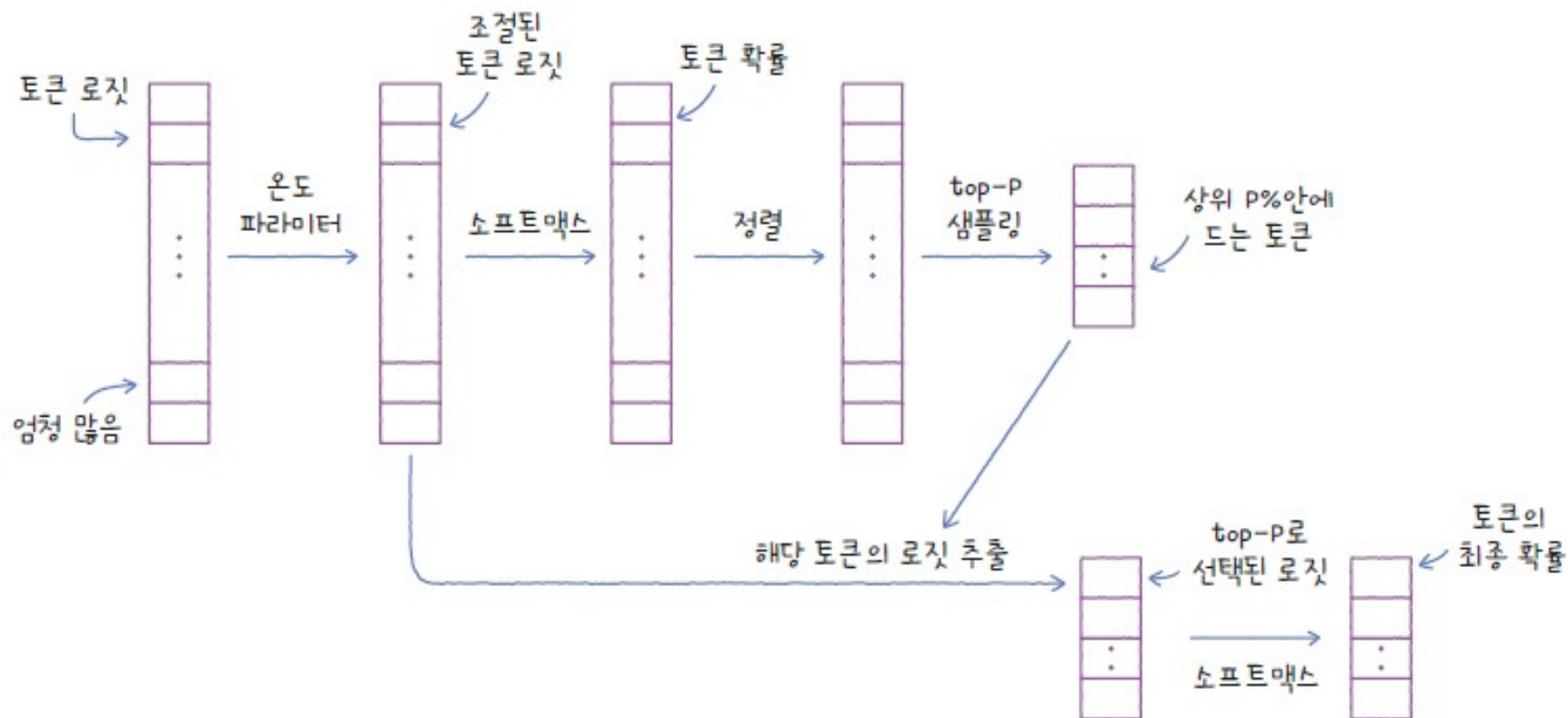
안녕하세요! 다이어리에 내년의 공휴일 정보가 포함되어 있는지 정확히 알려드리기 위해서는 제품의 최신 모델이나 디자인을 확인해야 합니다. 현재로서는 직접 확인하실 수 없으시다면, 저희가 가장 빠르게 답변 드리기 위해 제품 담당자에게 문의하시거나, 쇼핑몰 고객센터에 연락해 주시는 것이 좋을 것 같아요. 담당자께서 자세한 내용을 알려드릴 수 있을 거예요! 도와드리기 위해 최선을 다하겠습니다. 감사합니다.

- top-k 방식은 최상위 토큰 개수를 고정하기 때문에 비교적 높은 확률을 가진 토큰임에도 불구하고 선택에서 제외될 가능성이 있음
- 이에 반해 top-p 샘플링은 대상 토큰을 누적 확률로 지정하기 때문에 다양한 확률 분포에 유연하게 대처할 수 있음

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(38)

### ○ top-p 샘플링

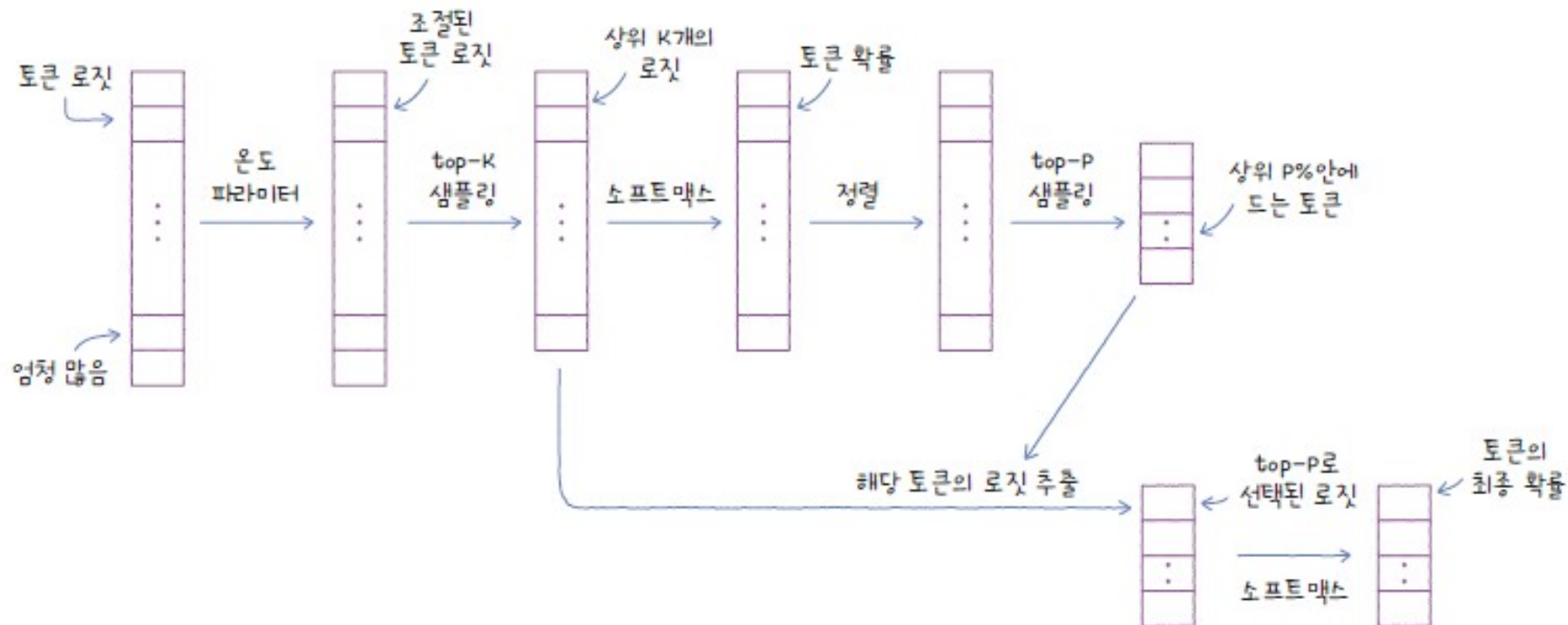
- top-p 샘플링을 하려면 소프트맥스 함수를 사용해 로짓을 확률로 바꿔야 함
  - 그다음 확률을 기준으로 토큰을 선택
  - 이 토큰들의 로짓을 다시 소프트맥스 함수에 통과시켜서 최종 토큰 확률을 계산



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(39)

### ○ top-p 샘플링

- top-p 샘플링의 단점 - 유용하지만 top-k 방식에 비해 계산량이 늘어남
  - 이런 경우 top-k와 함께 사용
  - 예) top-k로 먼저 최상위 로짓을 일부 선택한 다음 top-p 방식을 적용
  - transformers 패키지는 top-k와 top-p 방식을 동시에 사용할 수 있으며 그 중에 top-k가 로짓에 먼저 적용



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(40)

### ○ top-p 샘플링

- **top\_k 매개변수와 top\_p 매개변수를 함께 사용**

```
output = pipe(messages, max_new_tokens=200, return_full_text=False,  
               do_sample=True, temperature=2.0, top_k=100, top_p=0.9)  
print(output[0]['generated_text'])
```



안녕하세요! 다이어리에 내년도 공휴일이 포함되어 있는지 확인해 드리려면 제품 담당자에게 문의하시는 것이 가장 정확할 것 같아요. 담당자분께서는 최신 정보와 제품의 정확한 사양을 알려드리실 수 있을 거예요. 혹시 지금 바로 확인이 필요하시다면, 저희 웹사이트에 있는 고객 지원 섹션을 통해 문의해 보시는 건 어떨까요? 도움이 될 거예요!

- 위 코드에서는 temperature 매개변수로 로짓의 값을 먼저 조정한 후, top\_k 매개변수를 사용해 비교적 많은 개수의 토큰을 먼저 선택
- 그다음 top-p 샘플링을 적용



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(41)

## ○ 오픈AI 모델의 간략한 역사

- 오픈AI는 GPT(Generative Pre-trained Transformer)란 이름의 디코더 기반의 LLM을 개발
  - 2018년 GPT-1을 출시
  - 2019년 더 큰 모델인 GPT-2를 공개
  - 2020년 1,750억 개의 파라미터를 가진 GPT-3를 출시하면서 클로즈드 소스로 전환
  - 2022년 GPT-3.5와 ChatGPT가 등장하며 인공지능 기술이 본격적으로 주목받음
  - 2023년 텍스트뿐만 아니라 이미지를 처리할 수 있는 멀티모달(multi-modal) 모델인 GPT-4를 출시
  - 2024년 최신 모델인 GPT-4o 발표
- (학습) GPT-4o를 활용해 상품 문의에 대한 댓글을 생성

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(42)

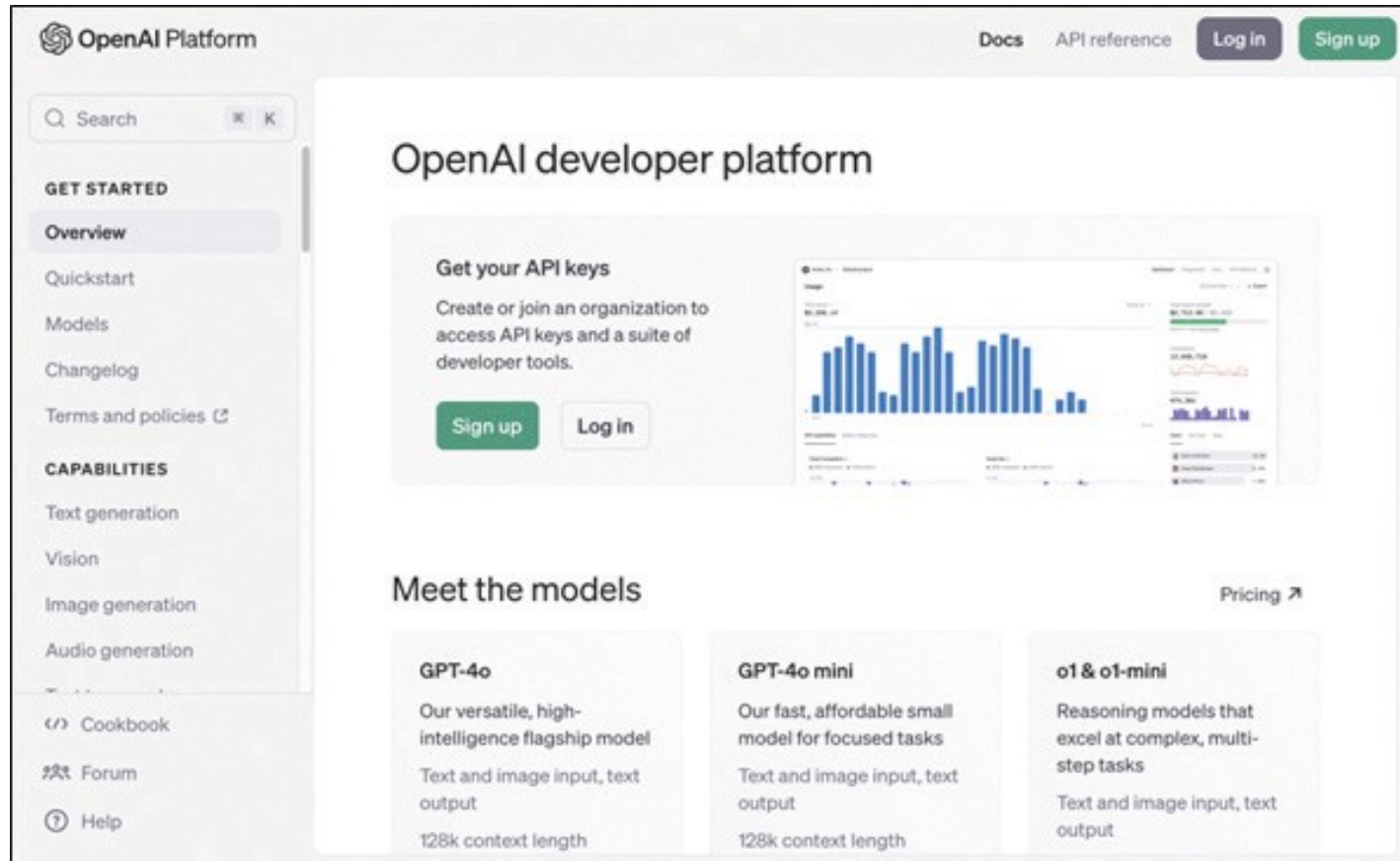
### ○ 오픈AI API 키 만들기

- 오픈AI에서 제공하는 모델을 사용하려면 먼저 오픈AI 사이트에 가입하고 API 키를 발급받아야 함
- 예전에는 신규 가입자에게 일정 양의 무료 크레딧을 제공했지만, 2024년 초부터 이 혜택이 중단
- 현재는 최소 \$5 정도를 충전해야 API를 사용할 수 있음
  - 이 금액이면 간단한 실험을 해보는 데 충분함

# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(43)

## ○ 오픈AI API 키 만들기

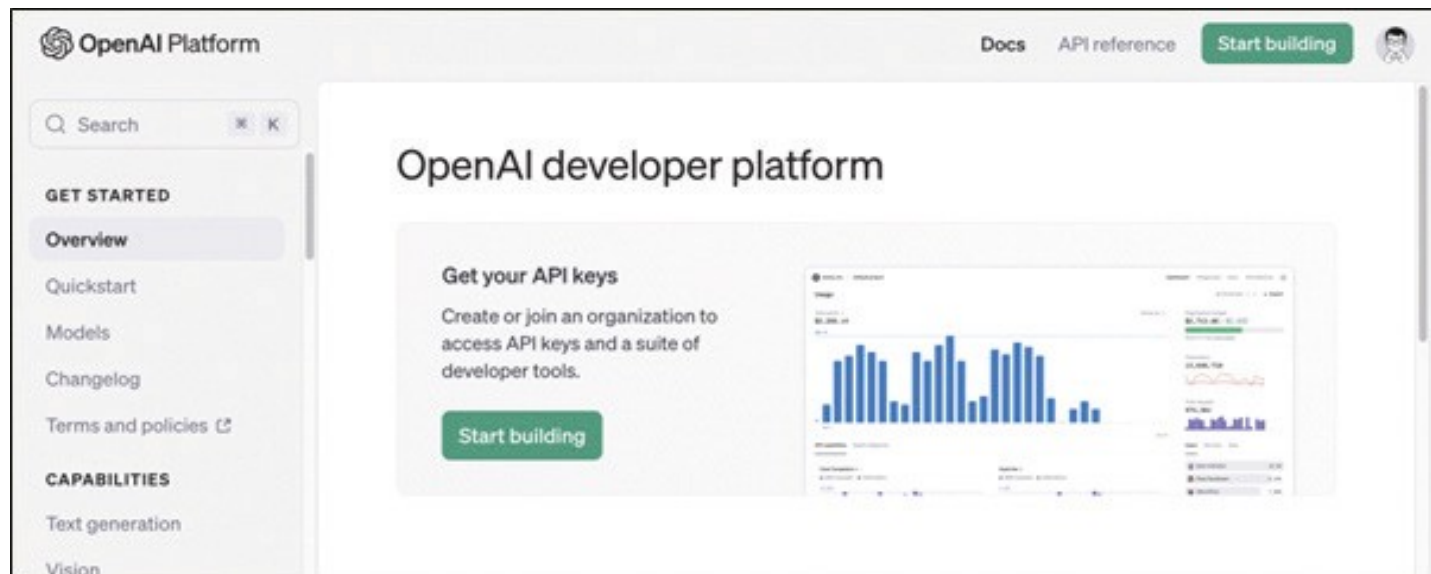
- 01      오픈AI API 사이트(<https://platform.openai.com/>)에 접속  
로그인 또는, 아직 회원이 아니라면 오른쪽 위에 있는 'Sign up' 버튼을 클릭하여 회원 가입을 진행



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(42)

## ○ 오픈AI API 키 만들기

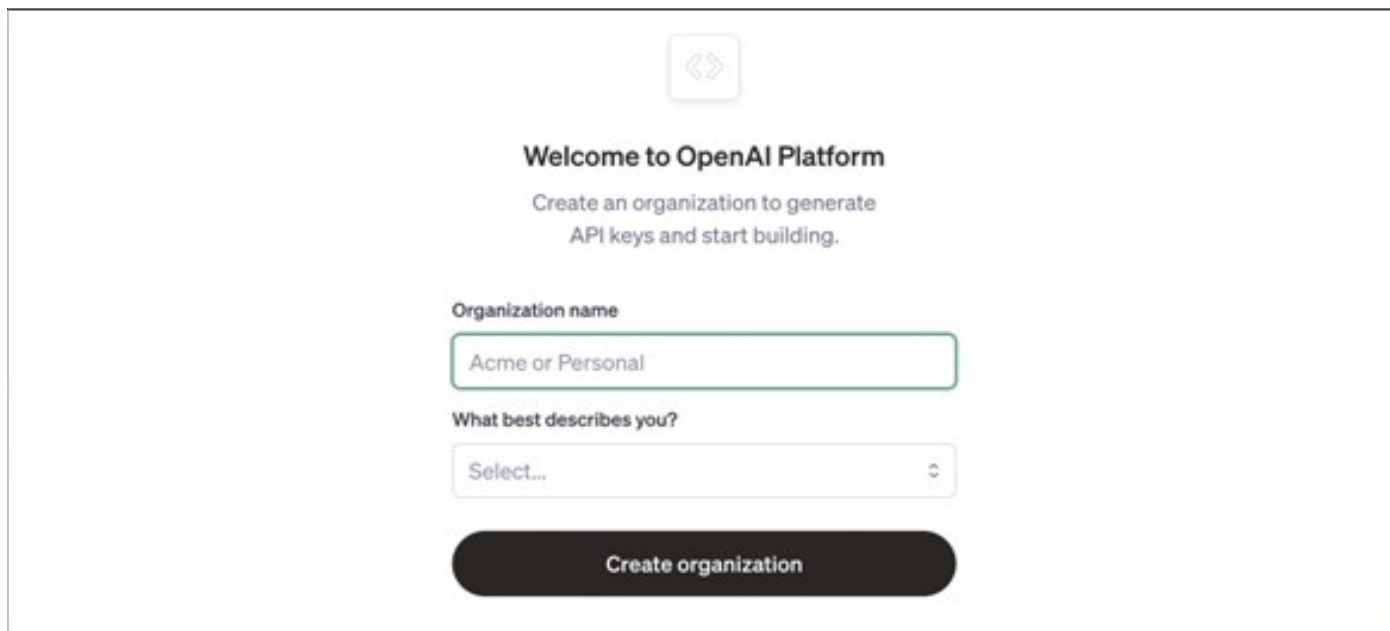
02 로그인 후 오른쪽 상단의 'Start building' 버튼을 클릭하여 기본 설정과 API 키를 발급받음



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(43)

## ○ 오픈AI API 키 만들기

- 03 'Start building' 버튼을 클릭하면 먼저 조직이나 개인의 이름을 입력하고, 기술에 어느 정도 익숙한지 질문에 적당한 답변을 입력 후 'Create organization' 버튼을 클릭  
이어지는 화면에서 팀원으로 초청하고 싶은 사람의 이메일을 적을 수 있음  
여기서는 'Continue' 버튼을 클릭해 이를 건너 뛰고 다음 화면으로 넘어감



The screenshot shows the 'Welcome to OpenAI Platform' page. At the top is a logo with two arrows pointing in opposite directions. Below the logo, the text reads 'Welcome to OpenAI Platform' followed by 'Create an organization to generate API keys and start building.' There are two input fields: 'Organization name' with the placeholder text 'Acme or Personal', and 'What best describes you?' with a dropdown menu showing 'Select...'. At the bottom is a large black button labeled 'Create organization'.

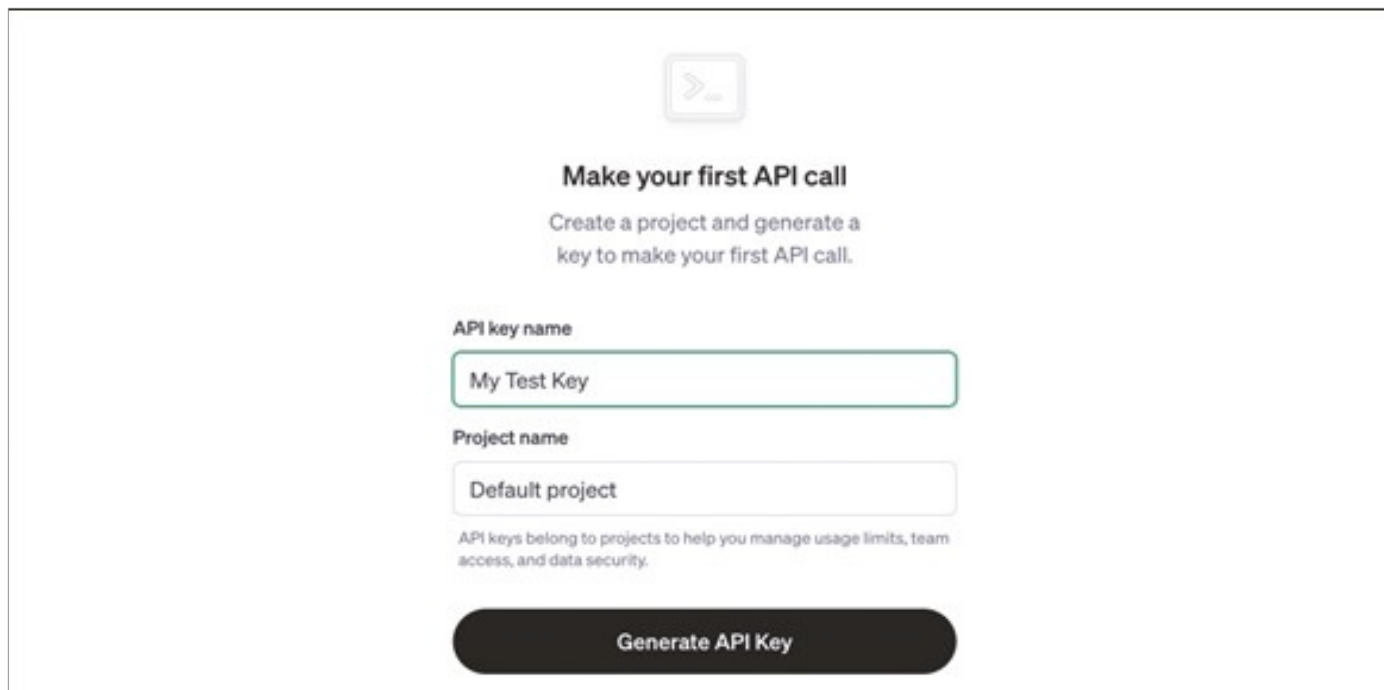
# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(44)

## ○ 오픈AI API 키 만들기

04 첫 번째 API 키를 발급받을 수 있음

여러 개의 키가 있을 때 구분하기 쉽도록 키 이름을 지정하거나 그냥 기본 값을 사용

'Generate API Key' 버튼을 클릭

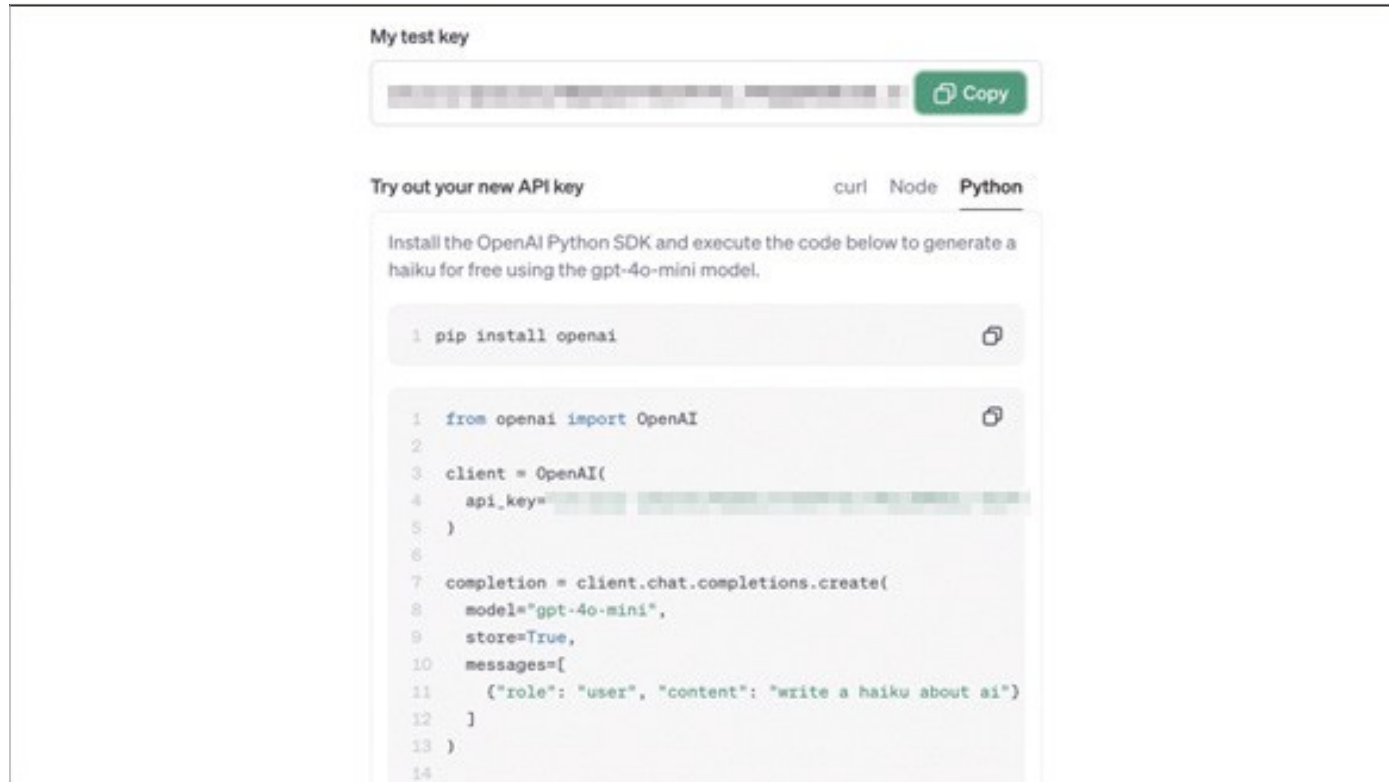


The screenshot shows the OpenAI API key generation page. At the top, there is a terminal icon and the heading "Make your first API call". Below this, a subheading reads "Create a project and generate a key to make your first API call." The form contains two input fields: "API key name" with the value "My Test Key" and "Project name" with the value "Default project". A note below the project name field states: "API keys belong to projects to help you manage usage limits, team access, and data security." At the bottom of the form is a large black button labeled "Generate API Key".

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(45)

### ○ 오픈AI API 키 만들기

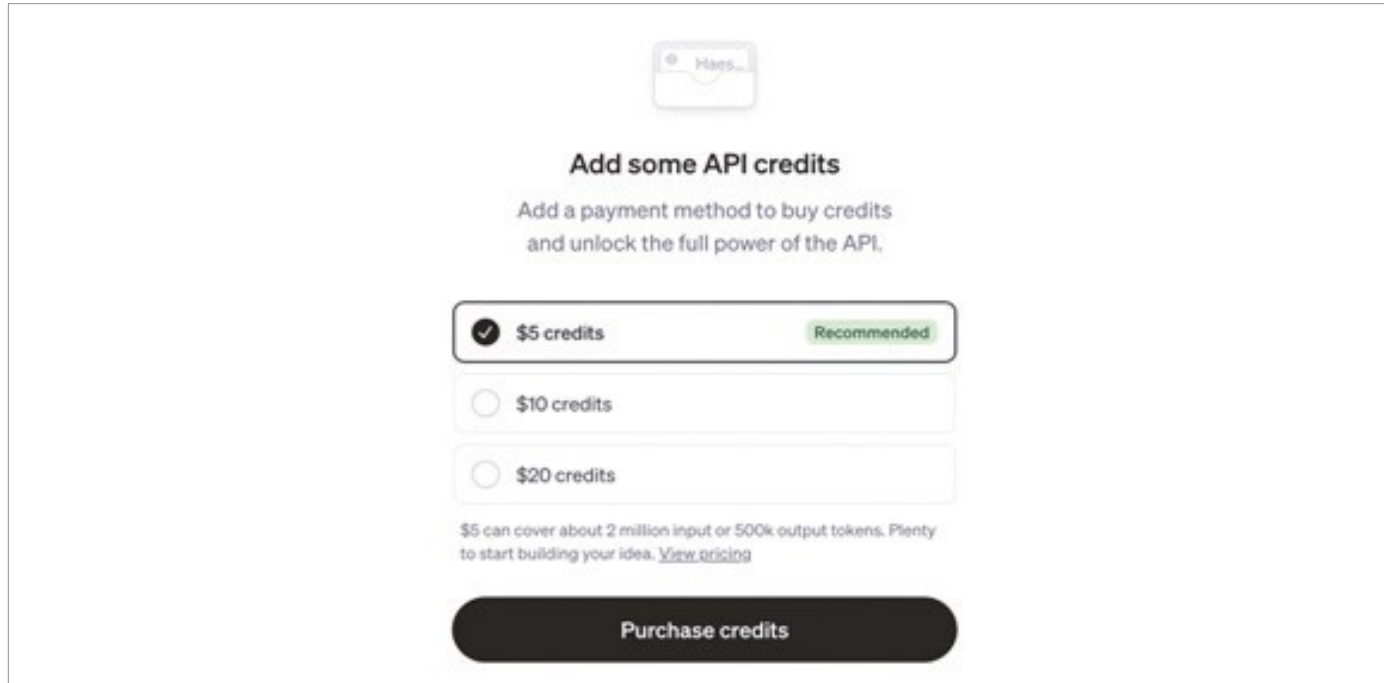
- 05 'Copy' 버튼을 클릭하여 키를 복사  
복사된 키를 메모장이나 텍스트 파일로 보관  
보안을 위해 API 키는 생성 시점에만 복사가 가능  
화면 아래 API 키를 사용하는 예제도 복사해 놓으면 좋음



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(46)

## ○ 오픈AI API 키 만들기

### 06 크레딧을 추가하고 신용카드 정보를 입력



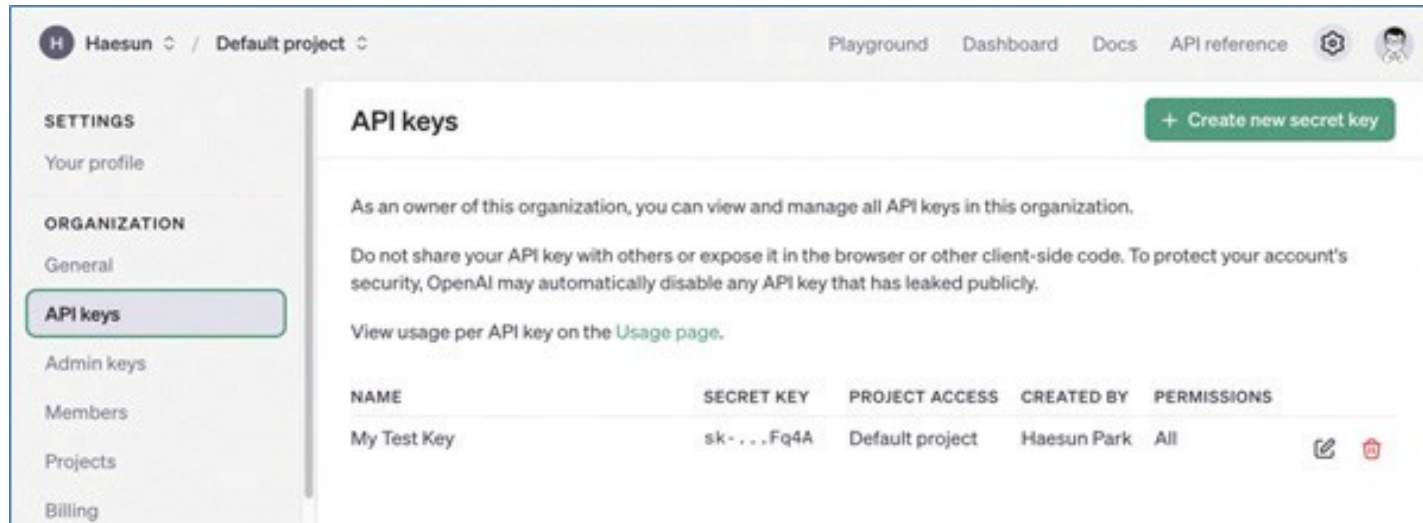
The screenshot shows the OpenAI API credits purchase interface. At the top, there is a small icon of a wallet with the name 'Hans...' next to it. Below the icon, the heading 'Add some API credits' is displayed. Underneath the heading, a subtext reads: 'Add a payment method to buy credits and unlock the full power of the API.' There are three radio button options for credit amounts: '\$5 credits' (which is selected and marked as 'Recommended'), '\$10 credits', and '\$20 credits'. Below these options, a small text block states: '\$5 can cover about 2 million input or 500k output tokens. Plenty to start building your idea. [View pricing](#)'. At the bottom of the form is a large black button labeled 'Purchase credits'.



# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(47)

## ○ 오픈AI API 키 만들기

- 07 로그인 후 오른쪽 상단의 톱니바퀴 아이콘을 누르면 설정 화면으로 이동  
여기에서 왼쪽에 있는 'API Keys'를 클릭하면 조금 전 발급받은 API 키를 확인  
(이전에 발급된 키를 다시 복사할 수는 없음) - 이 화면에서는 키의 이름을 수정, 삭제만 가능  
새로운 키를 만들고 싶다면, 오른쪽 상단의 'Create new secret key' 버튼을 클릭  
학습에서는 GPT-4o-mini를 사용



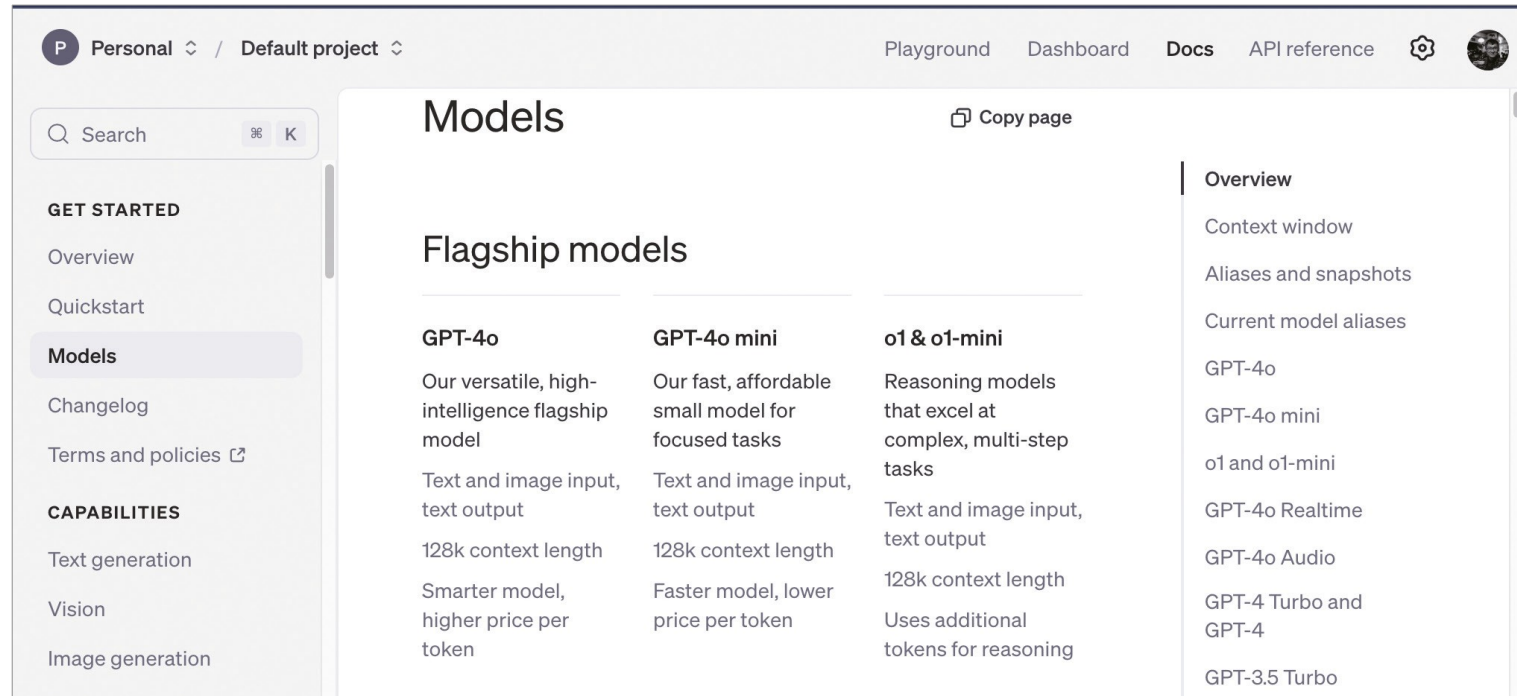
# SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(48)

## ○ 오픈AI API 키 만들기

### 08 오픈AI에서 제공하는 모델에 대한 정보 확인

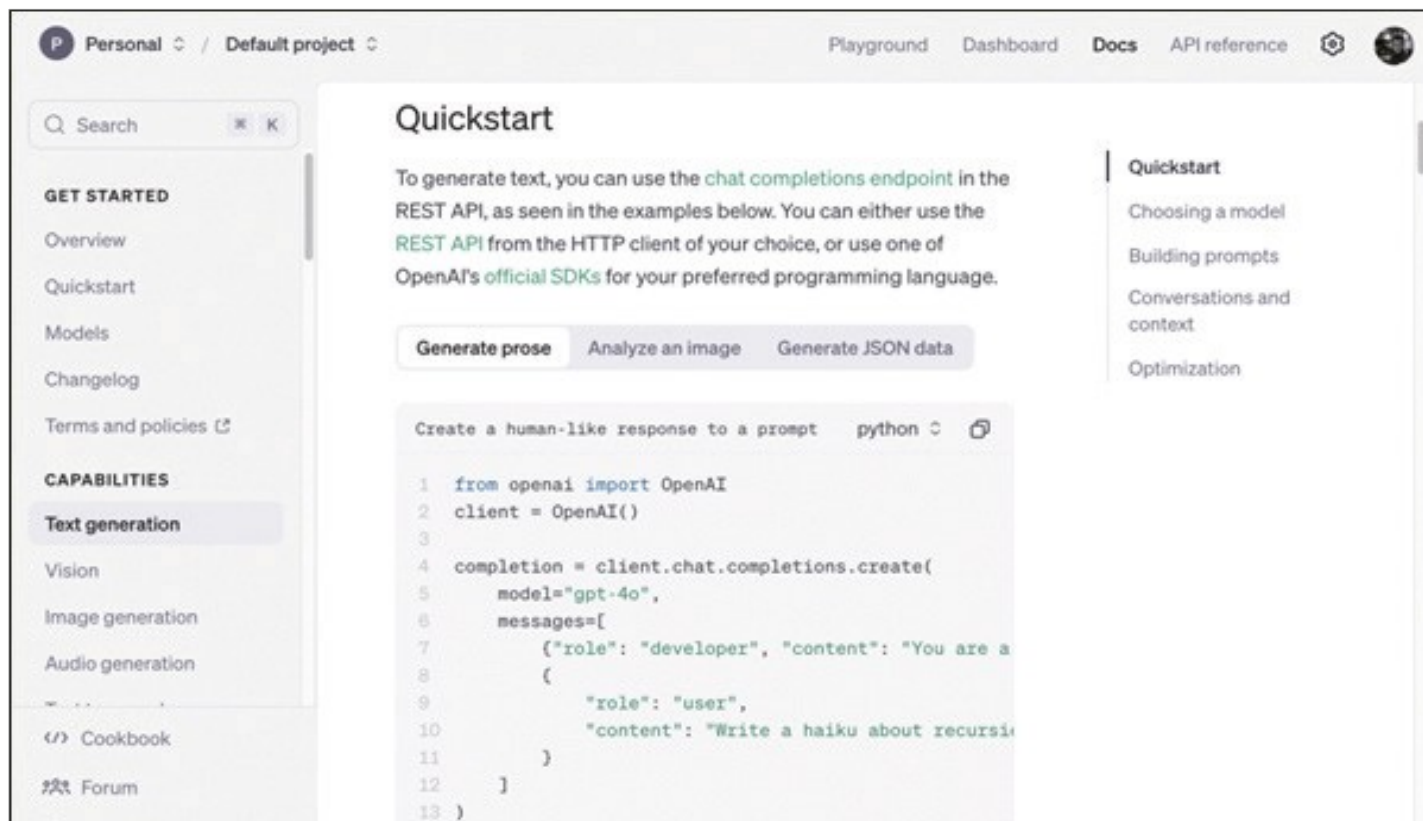
- 상단 메뉴 'Docs'를 클릭하고 왼쪽 메뉴에서 'Models'를 선택

(<https://platform.openai.com/docs/models>)



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(49)

- Docs 페이지의 왼쪽에 있는 'Text generation' 메뉴를 클릭 - 샘플 코드 확인
  - 예제 코드에 있는 메시지 형식이 앞서 EXAONE에 전달했던 채팅 템플릿과 매우 유사함
  - 이 코드를 복사하여 코랩에서 gpt-4o-mini 모델을 호출해서 고객 문의에 대한 댓글을 생성



## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(50)

### ○ 오픈AI API로 상품 질문에 대한 대답 생성하기

- openai 패키지에서 OpenAI 클래스를 импорт

```
from openai import OpenAI
```

- 이전에 생성한 API key를 사용해 클라이언트 객체를 생성

```
client = OpenAI(api_key="sk-proj-N7tW...43jMoA") # 여러분의 키를 입력하세요.
```

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(51)

### ○ 오픈AI API로 상품 질문에 대한 대답 생성하기

#### - 여기서 사용할 API는 가장 기본이 되는 채팅 완성(chat completion)

- client 객체의 chat.completions.create ( ) 메서드를 사용해 이 API를 호출
  - 이 때 model 매개변수에 사용할 모델을 지정하고 messages 매개변수에 채팅 메시지를 입력
  - 메시지는 앞서 만든 채팅 템플릿과 동일하므로 이를 그대로 사용

```
completion = client.chat.completions.create(  
    model="gpt-4o-mini",  
    messages=messages  
)
```

- 기본적으로 하나의 응답이 반환되므로 첫 번째 원소의 message.content 속성을 출력

```
print(completion.choices[0].message.content)
```

→

안녕하세요! 제품 담당자가 정확한 정보를 확인한 후에 답변드릴 수 있도록 시간이 필요합니다. 조금만 기다려 주시면 감사하겠습니다!

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(52)

### ○ 오픈AI API로 상품 질문에 대한 대답 생성하기

- 이 글을 작성하는 시점에 OpenAI는 top-k 샘플링을 지원하지 않음
- 대신 top-p 샘플링을 위한 top\_p 매개변수를 제공
  - top\_p 매개변수의 기본값은 1로 전체 토큰을 사용
    - 이를 0.9정도로 낮추면 높은 확률을 가진 토큰이 선택될 가능성이 조금 더 올라감

```
completion = client.chat.completions.create(  
    model="gpt-4o-mini",  
    messages=messages,  
    top_p=0.9  
)  
print(completion.choices[0].message.content)
```



안녕하세요! 해당 다이어리에 내년도 공휴일이 표시되어 있는지 확인하기 위해 제품 담당자에게 문의해보겠습니다. 조금만 기다려 주시면 감사하겠습니다!

## SECTION 10-3 대규모 언어 모델로 텍스트 생성하기(53)

### ○ 오픈AI API로 상품 질문에 대한 대답 생성하기

#### - temperature 매개변수로 온도 파라미터를 조정

- 오픈AI의 temperature 매개변수는 0~2 사이의 값을 지정해야 하며 기본값은 1
- 이 값을 높여서 조금 더 랜덤한 출력을 생성

```
completion = client.chat.completions.create(  
    model="gpt-4o-mini",  
    messages=messages,  
    temperature=1.8  
)  
print(completion.choices[0].message.content)
```



문의해 주셔서 감사합니다! 앞으로 내년도 공휴일이 다이어리에 포함되어 있는지에 대한 esclarecimento 사항은 제품 담당자에게 문의 드리겠습니다. 조금만 기다려 주시면 정확한 정보를 준비해드리겠습니다.

- temperature 값을 증가시키니 불안정한 답이 출력
- 오픈AI에서는 temperature 매개변수와 top\_p 매개변수를 동시에 사용하는 것보다 둘 중 하나를 선택해서 사용하도록 권장

# SECTION 10-3 마무리(1)

## ○ 키워드로 끝나는 핵심 포인트

- EXAONE
  - LG AI연구원에서 만든 트랜스포머 디코더 기반의 대규모 언어 모델
- 토큰 디코딩
  - 대규모 언어 모델이 출력한 로짓을 바탕으로 다음 토큰을 선택하는 과정
- GPT
  - 오픈AI에서 개발한 트랜스포머 디코더 기반의 대규모 언어 모델



## SECTION 10-3 마무리(2)

### ○ 핵심 패키지와 함수

#### - transformers

- AutoTokenizer - 허깅페이스에서 제공하는 사전 훈련된 LLM 모델의 토큰라이저를 직접 로드하기 위한 클래스
- 파이프라인 객체를 호출할 때의 매개변수
  - max\_new\_tokens 매개변수 - 모델이 생성할 최대 토큰 개수를 설정
  - return\_full\_text 매개변수를 False로 지정하면 모델이 생성한 텍스트만 반환
    - 기본값은 True
  - do\_sample 매개변수를 True로 지정하면 토큰 확률을 기반으로 다음 토큰을 선택
    - 기본값은 False
  - temperature 매개변수 - 모델이 출력한 로짓의 분포를 조정하는 온도 파라미터
  - top\_k 매개변수 - 가장 큰 확률을 가진 토큰 k개를 다음 토큰의 후보로 설정
    - 기본값은 50이며 이런 디코딩 전략을 top-k 샘플링이라 함
  - top\_p 매개변수 - 1.0보다 작게 설정하면 확률 크기 순으로 토큰을 나열했을 때 누적 확률이 지정한 값을 넘기지 않을 때까지 후보 토큰으로 설정
    - 기본값은 1.0이며 이런 디코딩 전략을 top-p 샘플링이라고 함

## SECTION 10-3 마무리(3)

### ○ 핵심 패키지와 함수

#### - openai

- OpenAI 클래스는 오픈AI의 API 호출을 위한 클라이언트 객체를 생성
  - api\_key 매개변수에는 오픈AI에서 발급받은 API 키를 지정
    - 이 매개변수를 지정하지 않으면 OPENAI\_API\_KEY 환경 변수에 저장된 값을 이용
- OpenAI.chat.completion.create( ) 메서드 - 채팅 완성 API를 호출하고 GPT 모델의 응답을 반환
  - model 매개변수 - 사용할 모델 아이디를 지정
  - messages 매개변수 - 모델에게 전달할 대화 메시지를 입력
  - temperature 매개변수 - 0~2 사이의 온도 파라미터를 조정 / 기본값은 1
  - top\_p 매개변수 - top-p 샘플링을 설정 / 기본값은 1
  - max\_completion\_tokens 매개변수 - 모델이 생성할 최대 토큰 수를 지정

## SECTION 10-3 확인 문제(1)

1 다음 중 EXAONE-3.5에서 사용하는 기술은?

- ① 멀티 헤드 어텐션
- ② 멀티 쿼리 어텐션
- ③ 그룹 쿼리 어텐션
- ④ 크로스 어텐션

2 다음 중 트랜스포머 디코더 기반의 대규모 언어 모델이 아닌 것은?

- ① BART
- ② Gemini
- ③ Claude
- ④ EXAONE

## SECTION 10-3 확인 문제(2)

3 다음 중 LLM에서 토큰을 생성할 때 널리 사용하는 디코딩 전략이 아닌 것은?

- ① top-k 샘플링
- ② top-p 샘플링
- ③ 그리디 서치
- ④ 이진 검색