# 자연어 처리

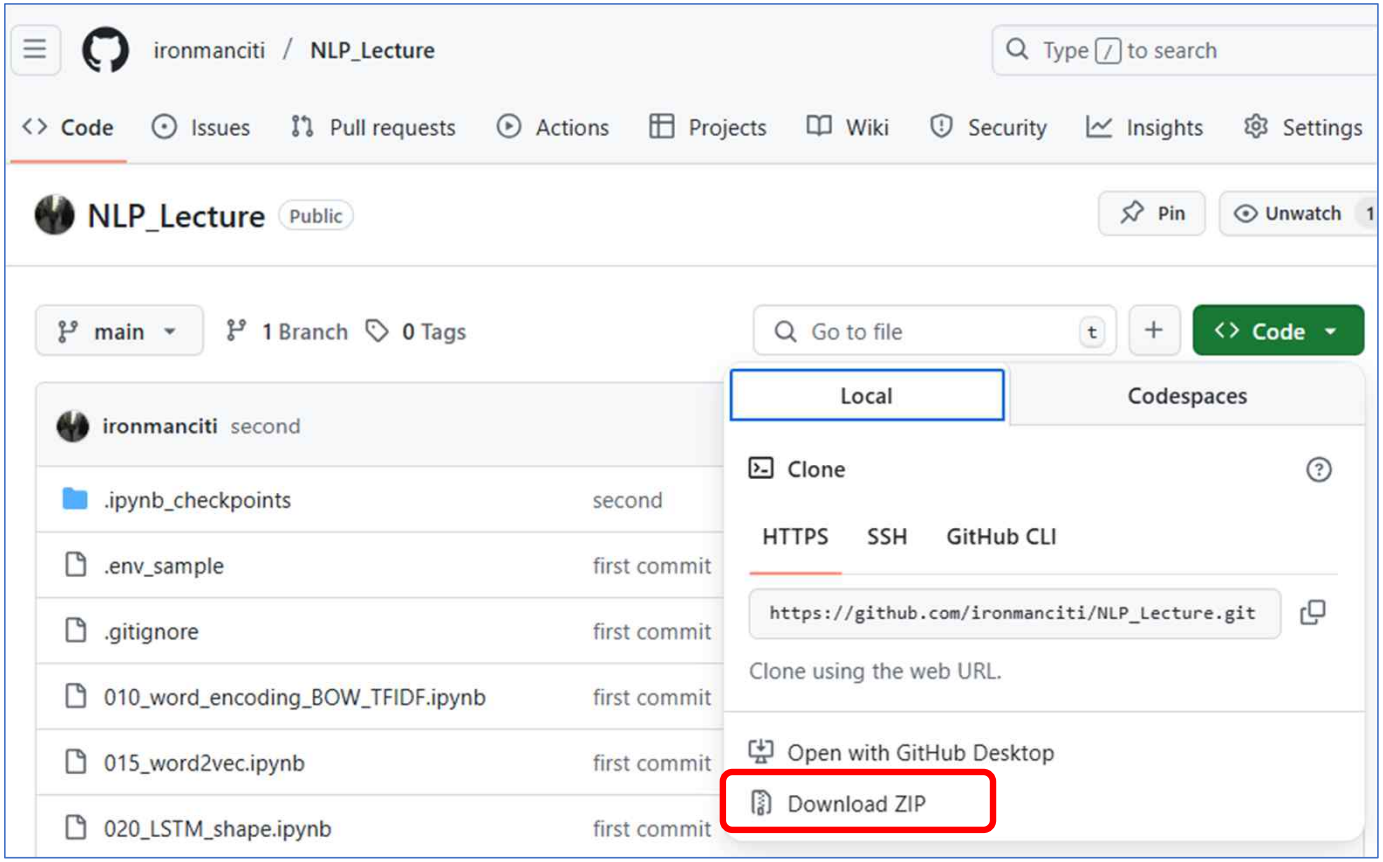
강사: 오영제

# 교육 환경 준비

- 인터넷 연결
- Chrome Browser
- Gmail ID for Colab

# Github Repository 에서 실습 code download

https://github.com/ironmanciti/NLP_Lecture

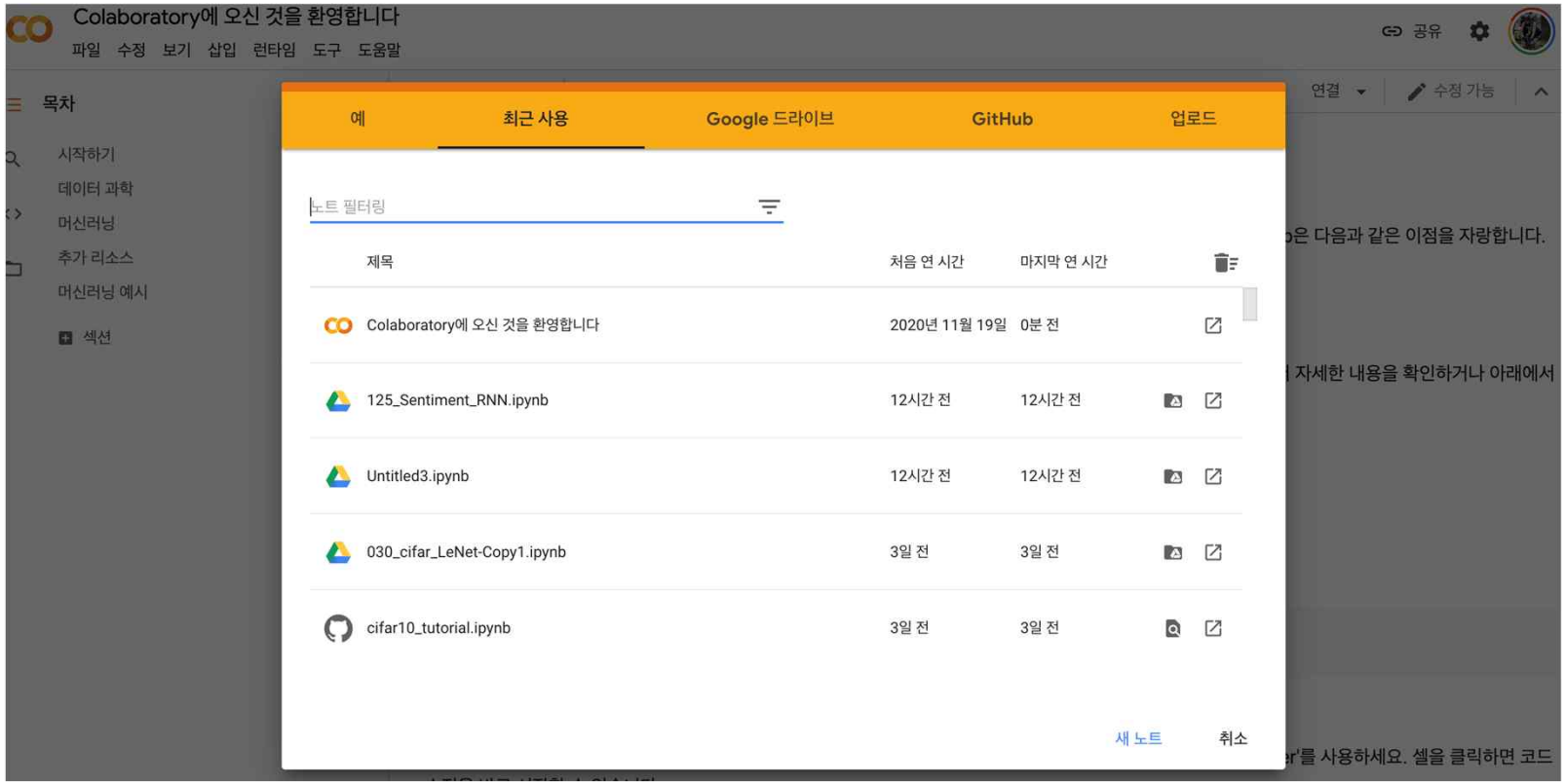


# Google Colaboratory 소개

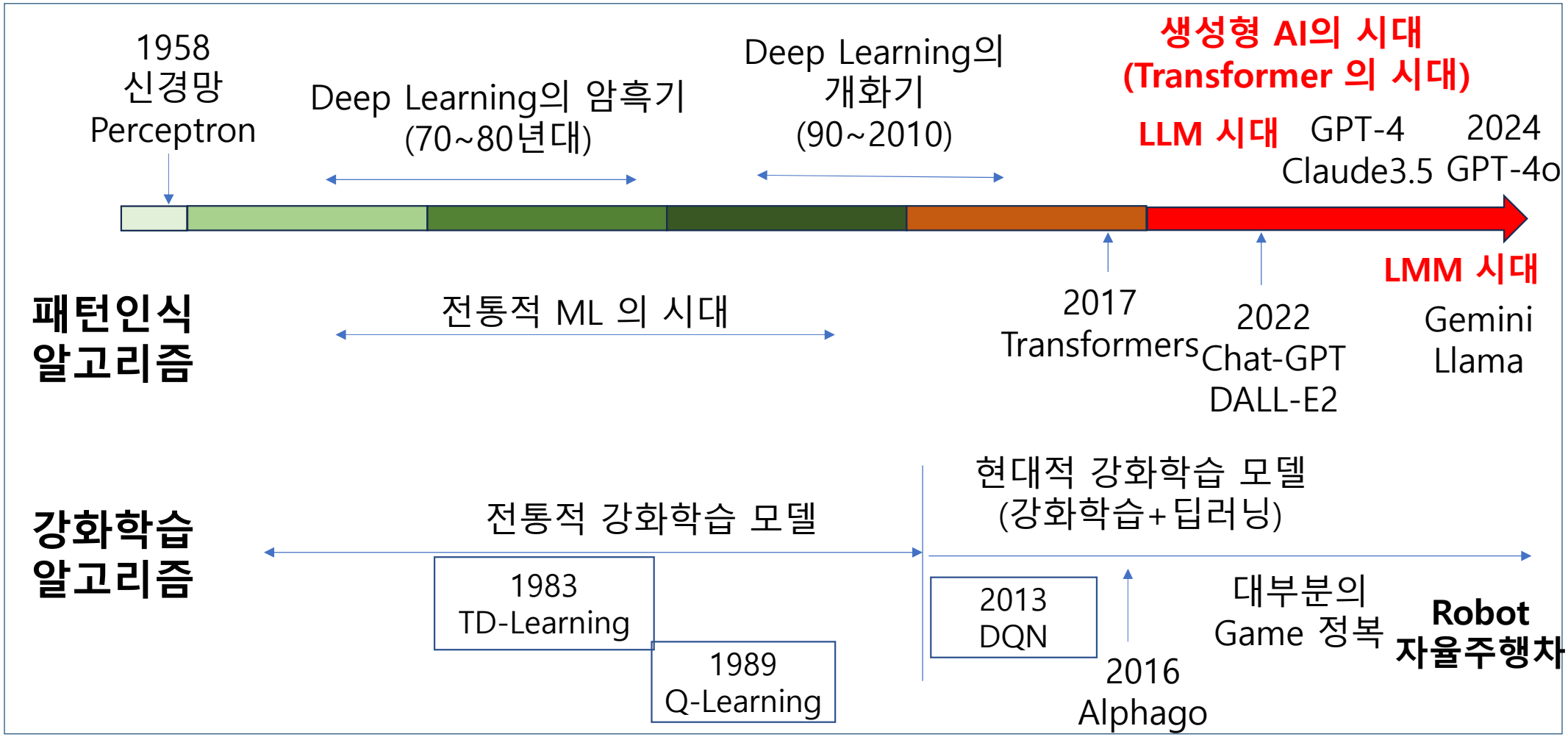
- Free GPU 제공
- Google Drive 와 연동
- Jupyter Notebook 환경
- Deep Learning beginner 를 위한 최적의 환경
- 각종 snippet 제공

# Google Colaboratory 사용하기

https://colab.research.google.com/



## AI 발전 History



# Natural Language Processing

# 자연어 처리(Natural Language Processing)

- 자연언어 : 일반 사회에서 자연히 발생한 언어
  - 한국어, 영어, 일본어 등
- 인공언어 : 프로그래밍 언어, 에스페란토어
- 자연어 처리
  - 자연어를 컴퓨터가 해독하고 그 의미를 이해하는 기술

# NLP 응용분야

Information Extraction


Sentimental Analysis

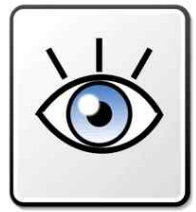


Image Captioning


Video Captioning

Advertisement Matching



Speech Recognition

Chatbot

Text Generation

Text Summarization


Spell Checking

Keyword Search

Machine Translation


Topic Modeling


Q&A

# NLP 에 대한 접근 방법 (Language Modeling)

| 기존 방식 (Data-Driven Approach) | 최근 방식 (Neural Network) |
| --- | --- |
| • 2010 년대 이전 전통적 방식<br>• 언어 전문가에 의한 고품질 sample data 확보가 중요<br>• **규칙기반 (Rule-Based)**<br>  • 언어학을 기반으로 한 rule-based program<br>• **통계기반 (Statistical Machine Translation)**<br>  • 말뭉치 (Corpus) 를 기반으로 통계 모델 및 전통적 machine learning 적용 | • **Deep Learning 이용**<br>• Word Embedding 을 기반으로 전체 입력 문장 단위로 처리<br>• 어순, 단어의 의미, 문맥 파악 등을 스스로 학습<br>• **언어 전문가 불필요**<br>• Word Embedding<br>• Bidirectional Recurrent Neural Network<br>• Encoder-decoder (seq2seq) model<br>• Attention model<br>• Transformer model<br>• BERT/GPT-3, etc |

# 최근의 NLP 발전 과정

- 2014 – RNN based sequence-to-sequence model 제안 (조경현 외)
- 2017 – **Transformer** (Google 연구진, "Attention is all you need") 논문 발표
- 2018. 6 – OpenAI GPT (Generative Pre-Training)
- 2018.11 – Google's BERT (Bidirectional Encoder Representations from Transformers)
- 2019. 2 – OpenAI GPT-2
- 2019.11 – Google's T5 (Text-To-Text Transfer Transformer)
- 2020. 5 – OpenAI GPT-3 → 1750 억 개의 parameter size
- 2022. 11 – ChatGPT (chatting version of GPT-3)
- 2023 – GPT-4, BARD
- 2024 – GPT-4o, Gemini → **BIG, BIGGER, MORE BIGGER**

# Tokenization & Tokenizer

# NLP(자연어처리)의 기본 recipe

- Corpus (말뭉치, **collection of texts**)
  - 자연어 분석 작업을 위해 만든 샘플 문서 집합
  - 단순히 소설, 신문 등의 문서를 모아 놓은 것. 혹은 품사, 형태소 등의 보조적 의미를 추가하여 구조적인 형태로 정리해 놓은 것 포함

- 토큰 (token)
  - 자연어 문서를 분석하기 위해 긴 문자열을 작은 단위로 나눈 것
  - Tokenize
    - 문자열을 여러 개의 조각, 즉 여러 개의 Token(토큰, 단어)들로 쪼개는 것
    - 특수 token : <START>, <EOS>, <UNK>, <PAD>, etc

- Text / sentence (문장)
  - Sequence of words

- Words (단어, 한글의 경우 형태소)
  - Sequence of meaningful characters
  - Word 는 영어의 경우 space 나 구두점(, ;) 으로 구분
  - 한국어, 일본어의 경우는 간격 없이 사용해도 읽을 수 있는 특징
    → 한글 형태소 분석기 or Sub-word 기법 사용

- Stop-words : 불용어

# Tokenization – Token 분리 방법

- 입력 text 를 의미를 가진 덩어리 (Token - 단어, 문장, 구, 절 등) 로 분할하는 작업

- Python split() method 사용
- nltk.sent_tokenize (문장 분리)
  nltk.word_tokenize (단어 분리)
- tensorflow 제공 Tokenizer : 단순히 구둣점, space 로 단어 분리
- Hugging Face tokenizers : BPE, WordPiece, SentencePiece 등 제공

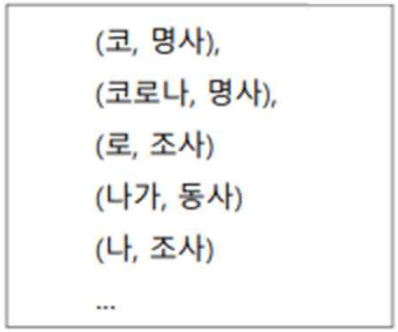
# 한글 자연어 처리

- 한국어 토큰화의 어려움
  - 조사가 띄어 쓰기 없이 바로 붙는다 (교착어) → 조사 분리 필요
  - 한국어는 띄어쓰기가 잘 지켜지지 않는다.
  - 한국어는 어순이 중요하지 않다.
- KoNLPy (Korean NLP in Python) – 코엔엘파이 (사전식)
  - Komoran, Mecab, Okt 등 내장
- 카카오 Khaiii

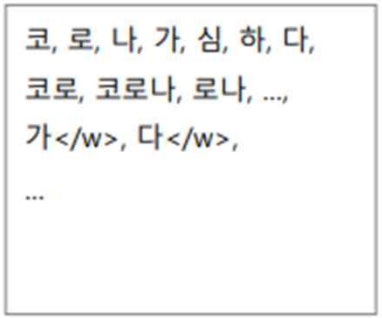
- Tokenization
  - text 를 word 또는 sub-word 로 분리하는 것

- Tokenizer
  - 사전 방식 – KoNLPy (Komoran, Mecab, Okt, etc) → 언어 지식 필요

  - sub-word 방식 – BPE, WordPiece, SentencePiece
    → 언어 지식 불필요, 다수의 언어에 공통으로 적용 가능



단어/형태소 사전



서브워드 사전

# Tokenizer 비교

| Tokenizer | 토큰 단위 | vocab size | 미등록단어에 대한 가정 |
|---|---|---|---|
| 사전 기반 | 알려진 단어/형태소<br>및 이들의 결합 | unlimited | • 알려진 단어/형태소의 결합이라 가정<br>• 필요시 형태소 분석 (형태 변형 가능)<br>• 빈번하지 않은 단어를 사전에 등록해 두면 잘 인식<br>• 사전에 등록되지 않은 단어는 UNK 처리 |
| sub-word | 알려진 글자 및<br>subword | fixed | • 알려진 sub-words 로 분해<br>ex) appear → app + ear<br>• 자주 등장하는 단어를 제대로 인식할 가능성 높음<br>• 알려진 글자로 분해하여 UNK 의 개수 최소화 |

- 각각의 장단점 있으므로 목적에 맞게 사용

# Tokenization 방법

- rule-based tokenization (공백 또는 구둣점으로 분리)
  - 문제점 : very big vocabulary 생성 (ex. Transformer XL : 267,735) →
    large embedding matrix 생성 → memory, time complexity 증가
- Subword tokenization
  - 원칙 : 빈번히 사용되는 단어는 더 작은 subword로 나뉘어 지면 안된다.
    가끔 사용되는 단어는 의미 있는 subword로 나뉘어 져야 한다.

    → 교착어 (한국어, 터키어, 일본어 등)의 token 화에 유용
    Bert 104 개국어 version 은 110,000 vocabulary size

# WPM(Word Piece Model) 개요

- 하나의 단어를 내부단어(subword)로 통계에 기반하여 띄어쓰기로 분리.
- 2015 년 처음 제안되어 Google 번역기에서 사용 (2016)
- 하나의 단어는 의미 있는 여러 단어들의 조합으로 구성된 경우가 많기 때문에, 단어를 여러 단어로 분리하여 보겠다는 전 처리 작업.
- 입력 문장에서 띄어쓰기는 언더바 (_)로 치환

- 기존의 띄어쓰기를 언더바(_)로 치환하는 이유는 차후 다시 문장 복원을 위한 장치.

  Ex) Jet makers feud over seat width with big orders at stake

  ➔ _J et _makers _fe ud _over _seat _width _with _big _orders _at _stake

- WPM 은 BPE (Byte Pair Encoding) 알고리즘 사용
  - 1994년에 제안된 데이터 압축 알고리즘
  - 훈련 데이터에 있는 단어들을 모든 글자(chracters) 또는 유니코드(unicode) 단위로 단어 집합(vocabulary)를 만들고, 가장 많이 등장하는 유니그램을 하나의 유니그램으로 통합

# Google SentencePiece

- 사전 토큰화 작업 없이 단어 분리 토큰화를 수행하므로 **언어에 종속되지 않음**

em __bed __ding ➡ embedding

(First sub-word 외에는 __ 으로 시작)

- 영어권 언어나 한국어는 단어 분리를 시도했을 때 어느정도 의미 있는 단위로 나누는 것이 가능
- Open Source 로 개방하여 실무에 사용 가능 (**한국어에 적용 가능**)
- SentencePiece 는 Unigram 알고리즘 사용

# 실습 : 037-Tokenizer Train 실습

- Keras tokenizer
- KoNLPy Okt tokenizer
- Google Sentencepiece
  - pip install sentencepiece
  - NSMC (Naver Sentiment Movie Corpus) data 를 이용한 tokenizer train
    - 아 더빙.. 진짜 짜증나네요 목소리
    - ['▁아', '▁더빙', '..', '▁진짜', '▁짜증나네요', '▁목소리']
    - [52, 752, 5, 25, 16020, 1401]

# 실습 : 040_tiktoken tokenizer 실습

- cl100k_base": "GPT-4, GPT-3.5-turbo, text-embedding-ada-002에서 사용
- 영어 문장: I love my dog
  - 토큰 ID: [40, 3021, 856, 5679]
  - 토큰 개수: 4
- 한글 문장: 코로나가 심하다
  - 토큰 ID: [168, 66391, 17835, 61415, 20565, 30027, 105, 16582, 13447]
  - 토큰 개수: 9

# 단어 (word) 와 문장의 숫자 표현

단어/문장 (string) 을 컴퓨터가 이해할 수 있는 숫자 (vector) 로 변환

How ?

**BOW**

**TF-IDF**

**Word Embedding**

# BOW (Bag of Words)

- 모든 문장을 토큰화 하고 각 문장에 토큰이 몇 번 등장하는지 count
- 각 token 을 feature 화 ➔ Text Vectorization

| | good | movie | not | a | did | like |
|---|---|---|---|---|---|---|
| good movie | 1 | 1 | 0 | 0 | 0 | 0 |
| not a good movie | 1 | 1 | 1 | 1 | 0 | 0 |
| did not like | 0 | 0 | 1 | 0 | 1 | 1 |

- Scikit-learn 의 nltk.CountVectorizer method 이용

# BOW (Bag of Words) 의 문제점

- 단어들 간의 순서를 유지할 수 없음

  → n-grams 기법으로 일부 해결

  bigrams 인접 단어의 쌍으로 구성 → ex) "is working" vs "not working"
  trigrams 인접 세개 단어로 구성 → ex) "not an issue"

- Counter 가 normalize 되어있지 않음

  → TF-IDF 로 해결

- 단순히 단어가 나타나는 횟수만 count 하므로 "**not** an issue, phone is working" 과 "an issue, phone is **not** working" 을 같은 의미로 간주

# n-grams : BOW 의 단어 순서 유지

- 1-gram : token 한 개
- 2-grams : token 두 개
- N-grams : token N 개

| | Corpus |
|---|---|
| Document 1 | good movie |
| Document 2 | not a good movie |
| Document 3 | did not like |

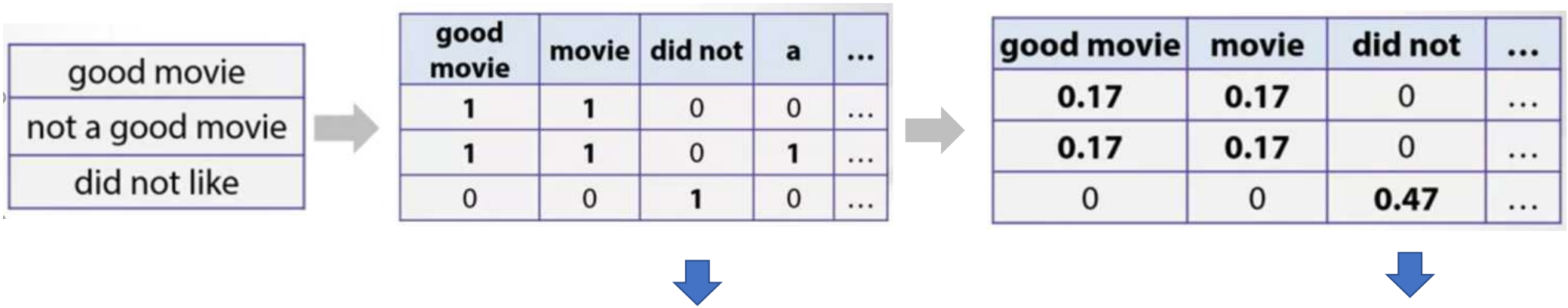
| good movie | movie | did not | a | … |
|---|---|---|---|---|
| 1 | 1 | 0 | 0 | … |
| 1 | 1 | 0 | 1 | … |
| 0 | 0 | 1 | 0 | … |

**문제점 : feature 개수가 기하급수적으로 증가**

# TF-IDF (Term Frequency - Inverse Document Frequency)

- TF(단어 빈도, term frequency)
  - 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값.
  - 이 값이 높을수록 특정 문서에서 중요하다고 간주
- DF(문서 빈도, document frequency)
  - 단어 자체가 전체 문서 집단 내에서 사용되는 빈도
  - DF 가 높으면 그 단어가 흔하게 등장한다는 것을 의미 (the, a, is, etc..)
- IDF(역문서 빈도, inverse document frequency) → DF 의 역수
- TF-IDF = TF * IDF
  - low TF-IDF - 전체 문서에서 공통으로 사용되는 단어임을 의미
  - high TF-IDF – 모든 문서가 아닌, 특정 문서에서 자주 사용되는 단어임을 의미

- TF-IDF 로 BOW 의 counter 를 대체
- row-wise normalize




**"did not" 2-gram 은 document 3 에만 나타나므로 TF-IDF가 High**

- Scikit-lean 의 TfidfVectorizer method 사용

# 실습: 010-문장의 Vector 표현

- sklearn BOW/TF-IDF 비교
- keras 제공 API 를 사용한 word encoding
- sentence 의 token 화 및 sequence 변환

# One-hot-encoding 표현의 문제점

- 단어를 단순히 index 번호에 따라 one-hot-encoding 으로 vectorize 하므로 **단어 간의 유사성**을 파악 못함

해결책 : **Word Embedding**

- **단어/문장 간 관련도 계산**
- **의미적/문법적 정보 함축**
- **전이 학습 가능**

# Word Embedding

- 숫자화된 단어의 나열로 부터 sentiment 추출

- 연관성 있는 단어들을 군집화하여 multi-dimension 공간에 vector 로 표시
  → **단어나 문장**을 **vector space** 로 끼워 넣음 (**embedding**)

- 예를 들어, 호감(positive), 비호감(negative) 두가지 label 에 따라 관련 단어들을 두개의 category 로 군집화

  ex) boring, bad, unfunny → negative

  funny, good, interesting → positive

# Word Embedding (Feature 화 표시)

| dimension | Man (5391) | Woman (9853) | King (4914) | Queen (7157) | Apple (456) | Orange (6257) |
|---|---|---|---|---|---|---|
| **성별** | -1 | 1 | -0.95 | 0.97 | 0.00 | 0.01 |
| **귀족** | 0.01 | 0.02 | 0.93 | 0.95 | -0.01 | 0.00 |
| **나이** | 0.03 | 0.02 | 0.7 | 0.69 | 0.03 | -0.02 |
| **음식** | 0.04 | 0.01 | 0.02 | 0.01 | 0.95 | 0.97 |

**Man (5931) 의 4 dimension vector 표시**

**King (4914) 의 4 dimension vector 표시**

# Embedding matrix (example)

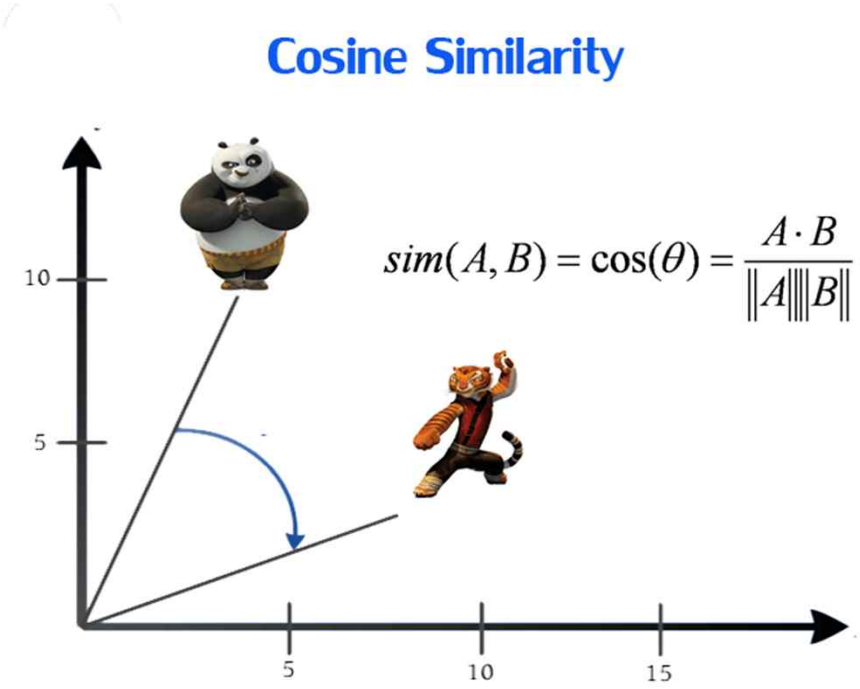
학습된 300 차원 featrues

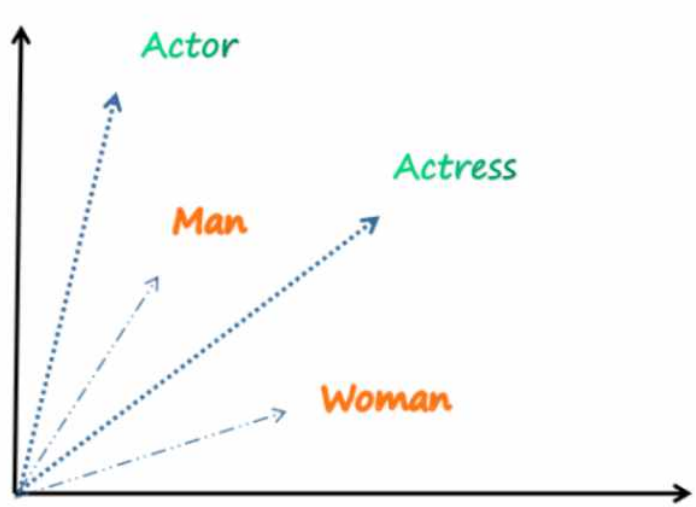
words

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 290 | 291 | 292 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **fox** | -0.348680 | -0.077720 | 0.177750 | -0.094953 | -0.452890 | 0.237790 | 0.209440 | 0.037886 | 0.035064 | 0.899010 | ... | -0.283050 | 0.270240 | -0.654800 | 0.10 |
| **ham** | -0.773320 | -0.282540 | 0.580760 | 0.841480 | 0.258540 | 0.585210 | -0.021890 | -0.463680 | 0.139070 | 0.658720 | ... | 0.464470 | 0.481400 | -0.829200 | 0.35 |
| **brown** | -0.374120 | -0.076264 | 0.109260 | 0.186620 | 0.029943 | 0.182700 | -0.631980 | 0.133060 | -0.128980 | 0.603430 | ... | -0.015404 | 0.392890 | -0.034826 | -0.72 |
| **beautiful** | 0.171200 | 0.534390 | -0.348540 | -0.097234 | 0.101800 | -0.170860 | 0.295650 | -0.041816 | -0.516550 | 2.117200 | ... | -0.285540 | 0.104670 | 0.126310 | 0.12 |
| **jumps** | -0.334840 | 0.215990 | -0.350440 | -0.260020 | 0.411070 | 0.154010 | -0.386110 | 0.206380 | 0.386700 | 1.460500 | ... | -0.107030 | -0.279480 | -0.186200 | -0.54 |
| **eggs** | -0.417810 | -0.035192 | -0.126150 | -0.215930 | -0.669740 | 0.513250 | -0.797090 | -0.068611 | 0.634660 | 1.256300 | ... | -0.232860 | -0.139740 | -0.681080 | -0.37 |
| **beans** | -0.423290 | -0.264500 | 0.200870 | 0.082187 | 0.066944 | 1.027600 | -0.989140 | -0.259950 | 0.145960 | 0.766450 | ... | 0.048760 | 0.351680 | -0.786260 | -0.36 |
| **sky** | 0.312550 | -0.303080 | 0.019587 | -0.354940 | 0.100180 | -0.141530 | -0.514270 | 0.886110 | -0.530540 | 1.556600 | ... | -0.667050 | 0.279110 | 0.500970 | -0.27 |
| **bacon** | -0.430730 | -0.016025 | 0.484620 | 0.101390 | -0.299200 | 0.761820 | -0.353130 | -0.325290 | 0.156730 | 0.873210 | ... | 0.304240 | 0.413440 | -0.540730 | -0.03 |
| **breakfast** | 0.073378 | 0.227670 | 0.208420 | -0.456790 | -0.078219 | 0.601960 | -0.024494 | -0.467980 | 0.054627 | 2.283700 | ... | 0.647710 | 0.373820 | 0.019931 | -0.03 |
| **toast** | 0.130740 | -0.193730 | 0.253270 | 0.090102 | -0.272580 | -0.030571 | 0.096945 | -0.115060 | 0.484000 | 0.848380 | ... | 0.142080 | 0.481910 | 0.045167 | 0.05 |
| **today** | -0.156570 | 0.594890 | -0.031445 | -0.077586 | 0.278630 | -0.509210 | -0.066350 | -0.081890 | -0.047986 | 2.803600 | ... | -0.326580 | -0.413380 | 0.367910 | -0.26 |
| **blue** | 0.129450 | 0.036518 | 0.032298 | -0.060034 | 0.399840 | -0.103020 | -0.507880 | 0.076630 | -0.422920 | 0.815730 | ... | -0.501280 | 0.169010 | 0.548250 | -0.31 |
| **green** | -0.072368 | 0.233200 | 0.137260 | -0.156630 | 0.248440 | 0.349870 | -0.241700 | -0.091426 | -0.530150 | 1.341300 | ... | -0.405170 | 0.243570 | 0.437300 | -0.46 |
| **kings** | 0.259230 | -0.854690 | 0.360010 | -0.642000 | 0.568530 | -0.321420 | 0.173250 | 0.133030 | -0.089720 | 1.528600 | ... | -0.470090 | 0.063743 | -0.545210 | -0.19 |
| **dog** | -0.057120 | 0.052685 | 0.003026 | -0.048517 | 0.007043 | 0.041856 | -0.024704 | -0.039783 | 0.009614 | 0.308416 | ... | 0.003257 | -0.036864 | -0.043878 | 0.00 |
| **sausages** | -0.174290 | -0.064869 | -0.046976 | 0.287420 | -0.128150 | 0.647630 | 0.056315 | -0.240440 | -0.025094 | 0.502220 | ... | 0.302240 | 0.195470 | -0.653980 | -0.29 |
| **lazy** | -0.353320 | -0.299710 | -0.176230 | -0.321940 | -0.385640 | 0.586110 | 0.411160 | -0.418680 | 0.073093 | 1.486500 | ... | 0.402310 | -0.038554 | -0.288670 | -0.24 |
| **love** | 0.139490 | 0.534530 | -0.252470 | -0.125650 | 0.048748 | 0.152440 | 0.199060 | -0.065970 | 0.128830 | 2.055900 | ... | -0.124380 | 0.178440 | -0.099469 | 0.00 |
| **quick** | -0.445630 | 0.191510 | -0.249210 | 0.465900 | 0.161950 | 0.212780 | -0.046480 | 0.021170 | 0.417660 | 1.686900 | ... | -0.329460 | 0.421860 | -0.039543 | 0.15 |

20 rows × 300 columns

# 유사도 측정 (Cosine Similarity)

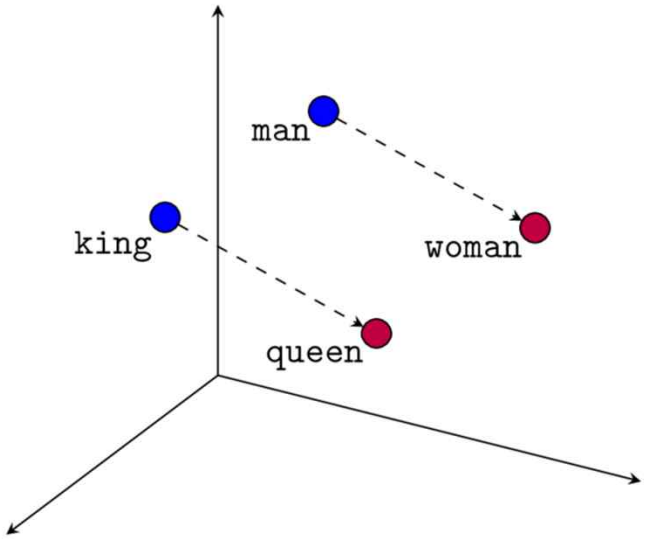


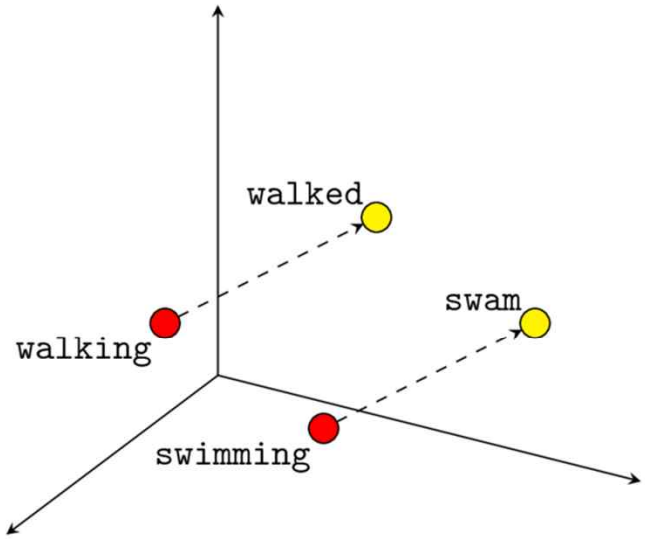

A, B 유사하면 ≈ 1
관련 없으면 ≈ 0

# Word2Vec (Word Embedding) 의 결과

king – queen ≈ man - woman




성별




시제


국가-수도

https://ronxin.github.io/wevi/

# 실습 : 017_embedding

- Embedding 모델을 이용하여 embedding 변환

```
sentences = [
    '나는 인공지능 공부를 좋아한다.',
    '인공지능은 매우 흥미롭다.',
    '오늘 날씨가 흐리고 비가 온다.'
]
```

→

| | 0 | 1 | 2 | 3 | 4 | 5 | \ |
|---|---|---|---|---|---|---|---|
| 나는 인공지능 공부를 좋아한다. | -0.021738 | -0.012001 | -0.013173 | 0.011548 | 0.049148 | -0.015339 | |
| 인공지능은 매우 흥미롭다. | -0.015396 | 0.010621 | -0.042270 | 0.030104 | 0.010643 | -0.035229 | |
| 오늘 날씨가 흐리고 비가 온다. | 0.019867 | -0.001774 | -0.074277 | -0.024289 | 0.029764 | 0.045755 | |

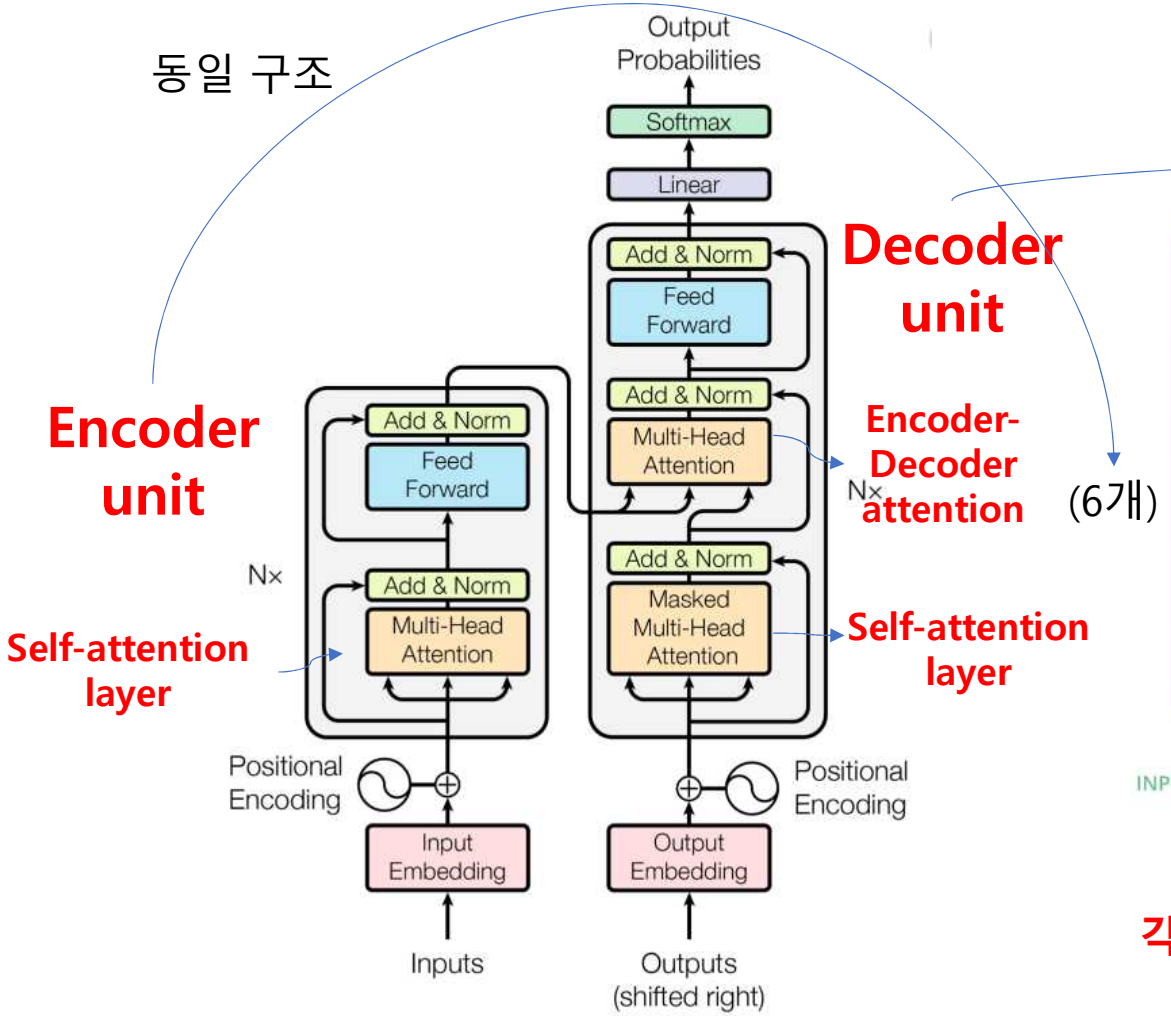
- 문장간의 cosine 유사도 계산

```
문장 간 코사인 유사도:
나는 인공지능 공부를 좋아한다. vs 인공지능은 매우 흥미롭다.: 0.5847
나는 인공지능 공부를 좋아한다. vs 오늘 날씨가 흐리고 비가 온다.: 0.1017
인공지능은 매우 흥미롭다. vs 오늘 날씨가 흐리고 비가 온다.: 0.1661
```

# Transformers

# Transformer Architecture

동일 구조




**Encoder unit**

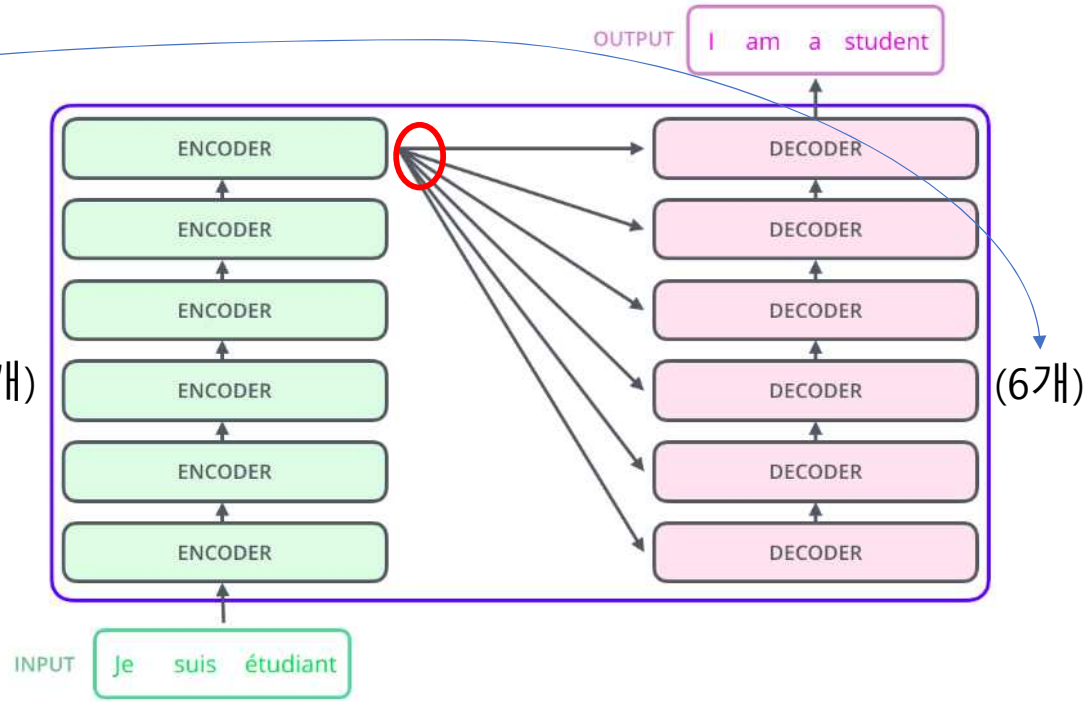
**Decoder unit**

**Encoder-Decoder attention**

**Self-attention layer**

**Self-attention layer**

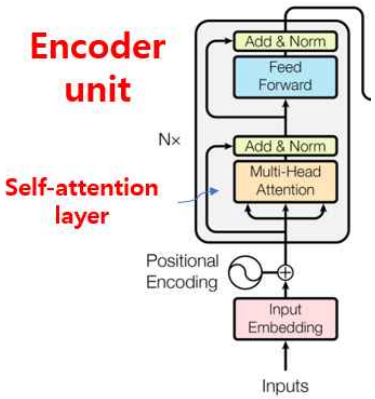
Last encoder 의 output 이 decoder-stack 의 모든 decoder 로 보내진다

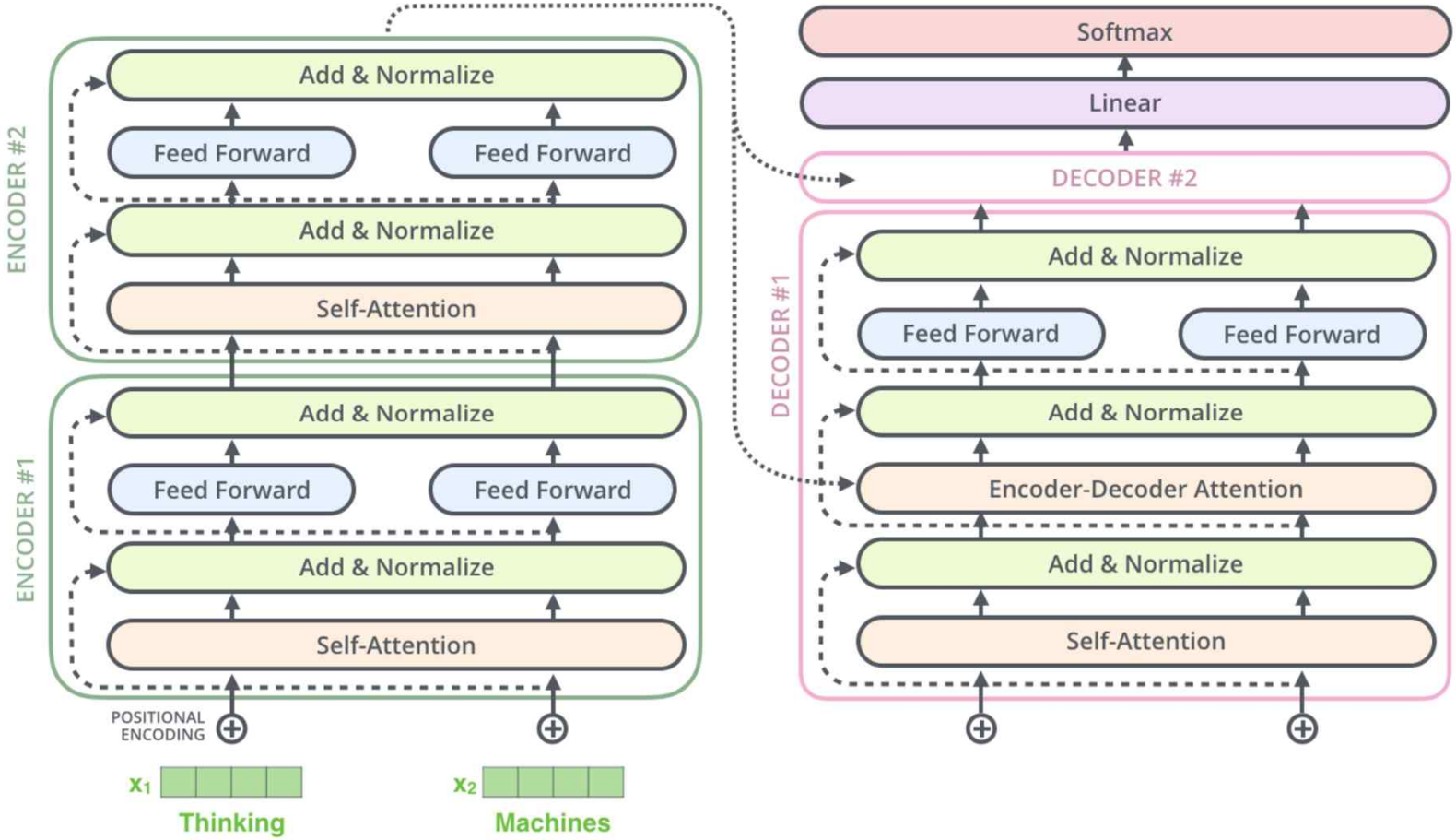



word embedding

**각 word 들을 병렬 처리**

반복 (6 개)


Encoder unit
Add & Norm
Feed Forward
Nx
Add & Norm
Multi-Head Attention
Self-attention layer
Positional Encoding
Input Embedding
Inputs


ENCODER #2
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
ENCODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Self-Attention
POSITIONAL ENCODING
x1
x2
Thinking
Machines
Softmax
Linear
DECODER #2
DECODER #1
Add & Normalize
Feed Forward
Feed Forward
Add & Normalize
Encoder-Decoder Attention
Add & Normalize
Self-Attention

# Self-Attention (Intra-Attention)

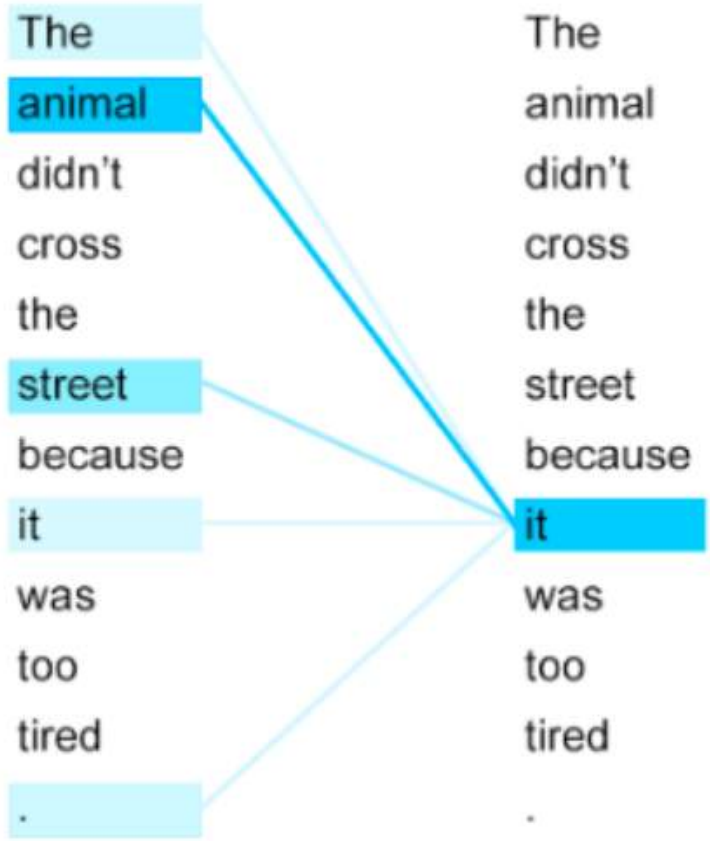
Attention 을 자기 자신에 대해 수행

→ 문장 내 단어들 간의 유사도를 구함

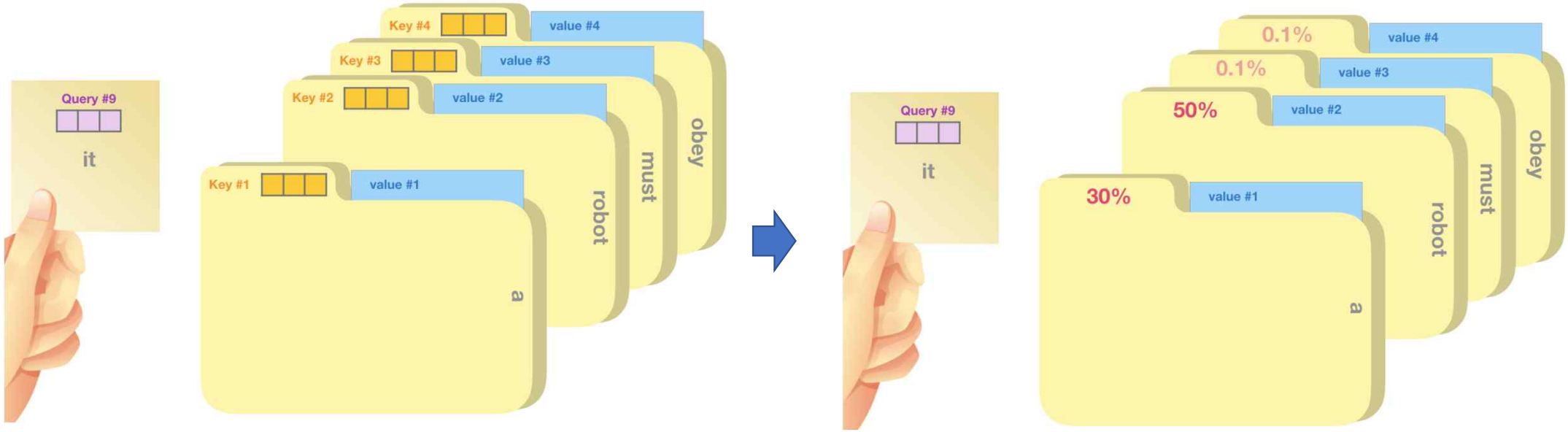
Self-attention 계산

3 개의 vector 필요 (훈련 과정에서 스스로 학습)

- Query Vector
- Key Vector
- Value Vector

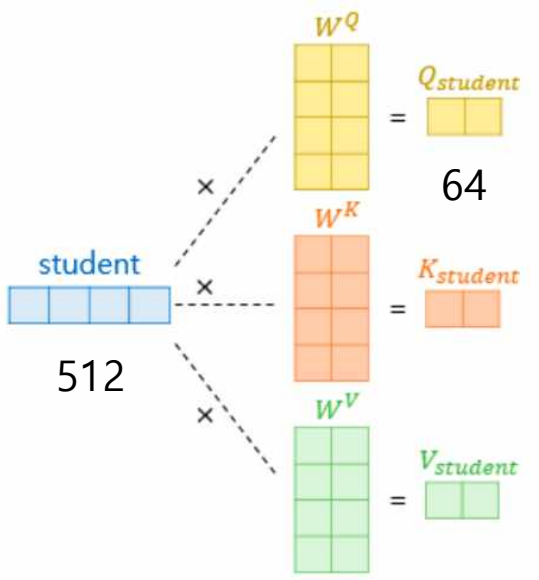


# Self-Attention 과 Query, Key, Values

# Self-Attention 계산 (1~4 단계로 구성)

- 1 단계 - Q, K, V vector (projection vector) 생성 단계
  각 단어 vector 를 $d_{model}$ / num_heads 차원의 vector 들로 변환 (512 / 8 = 64)

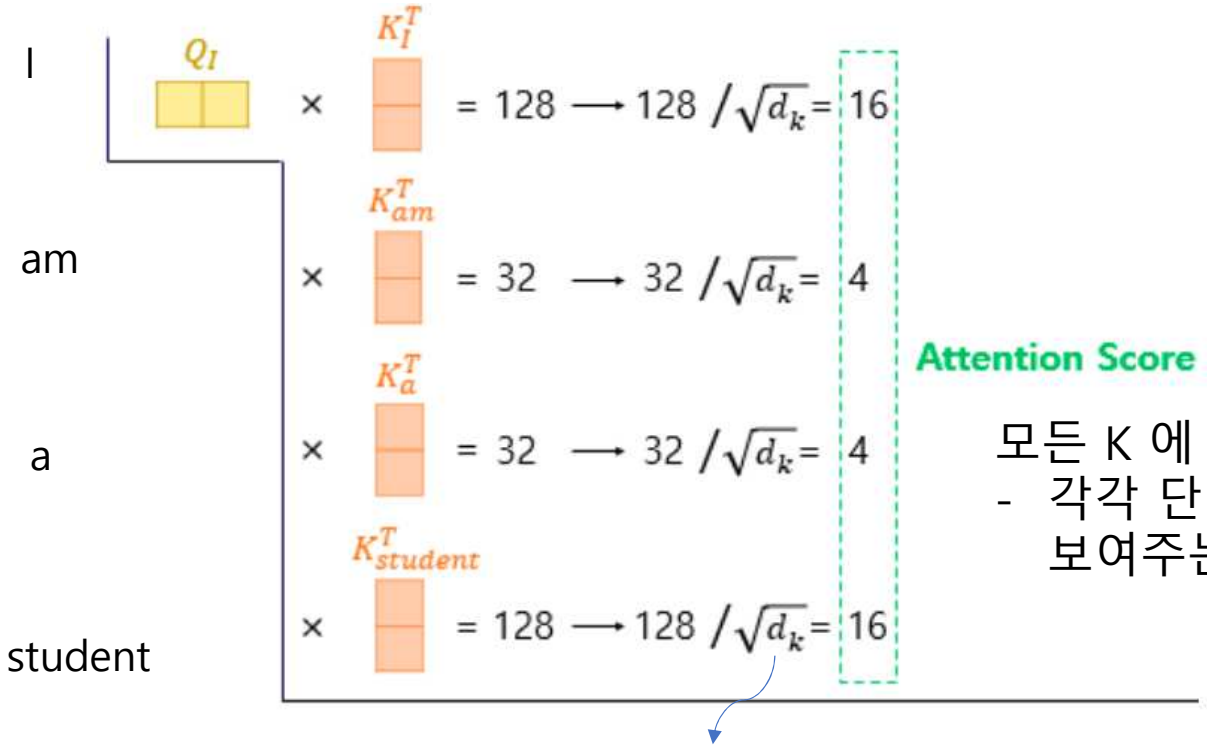



가중치 행렬 Wq, Wk, Wv 는 훈련과정에서 학습

```
# Linear Layer
self.wq = tf.keras.layers.Dense(d_model)
self.wk = tf.keras.layers.Dense(d_model)
self.wv = tf.keras.layers.Dense(d_model)
```

- 2~4 단계 – Q, K, V 를 이용한 scoring 단계

- 2 단계 - 모든 K vector 에 대하여 attention score 를 구함
  - Scaled dot product Attention : $q \cdot k/\sqrt{n}$
  - 특정 단어에 대한 input sequence 내 모든 단어의 score 필요(얼마나 focus 할지 결정)

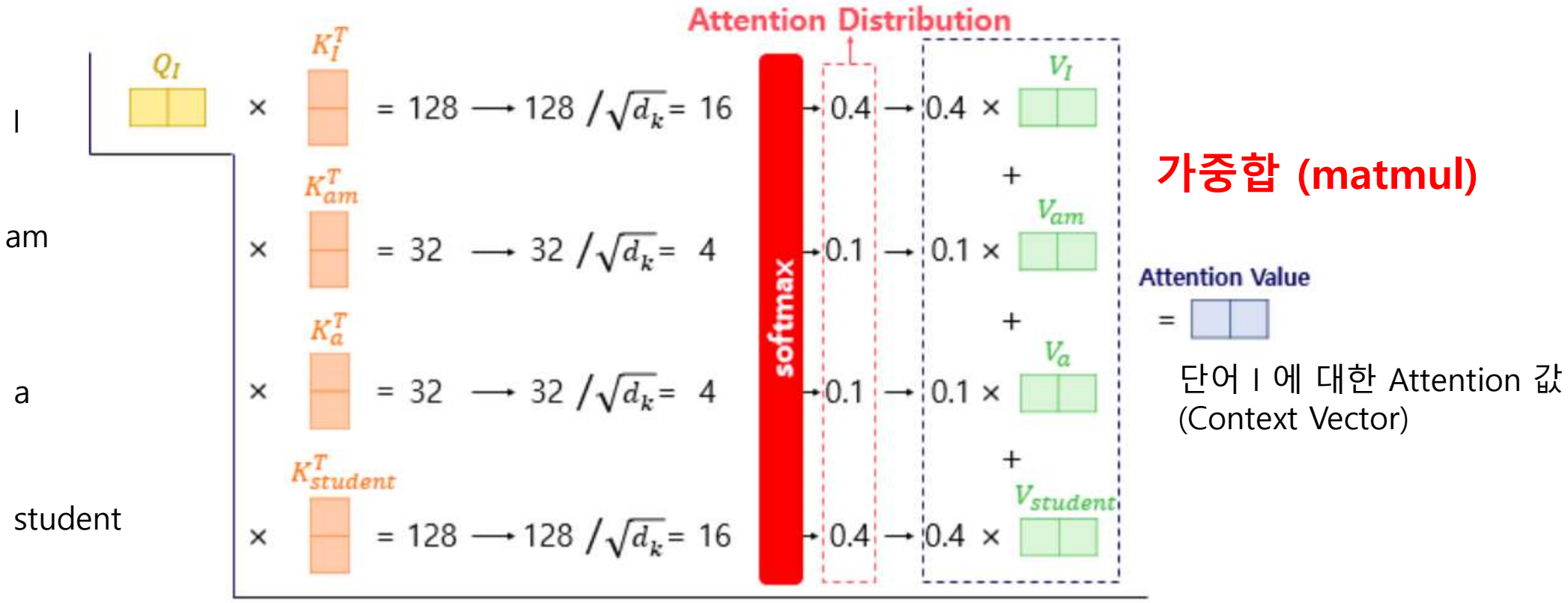



모든 K 에 대한 attention score
- 각각 단어가 I 와 얼마나 연관되어 있는지 보여주는 수치 (vector 의 similarity)

Dot product attention 에 scaling ($/\sqrt{n}$ ) 추가

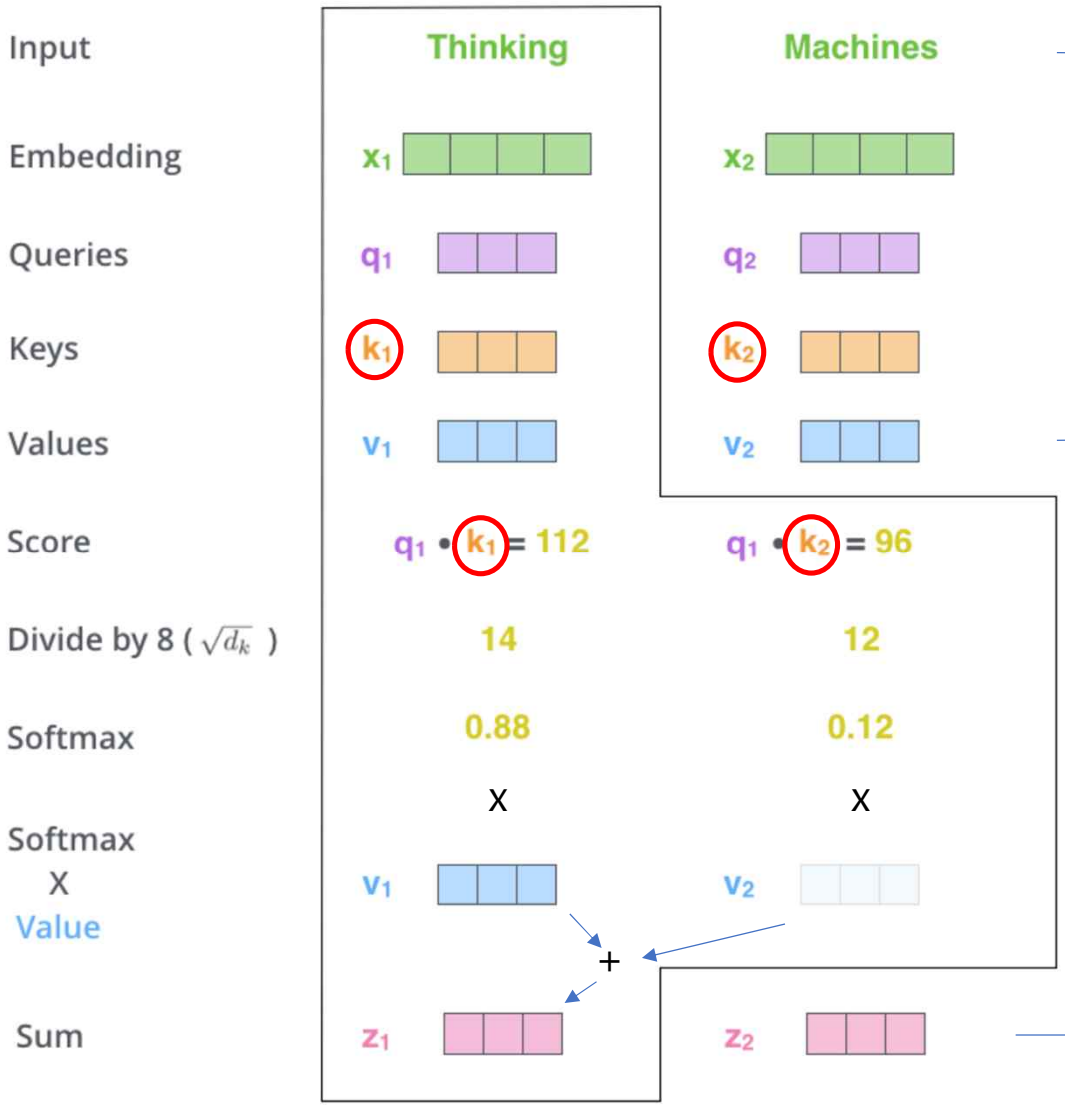
Scaling : $\sqrt{64}$ = 8

- 3 단계 – Softmax 로 Attention 분포를 구한 후
  4 단계 - 이를 이용하여 V vector 를 가중합 하여 Attention Value 를 구함



Self-Attention 계산

All-together




1단계

Dot product ≈ 유사도

2단계

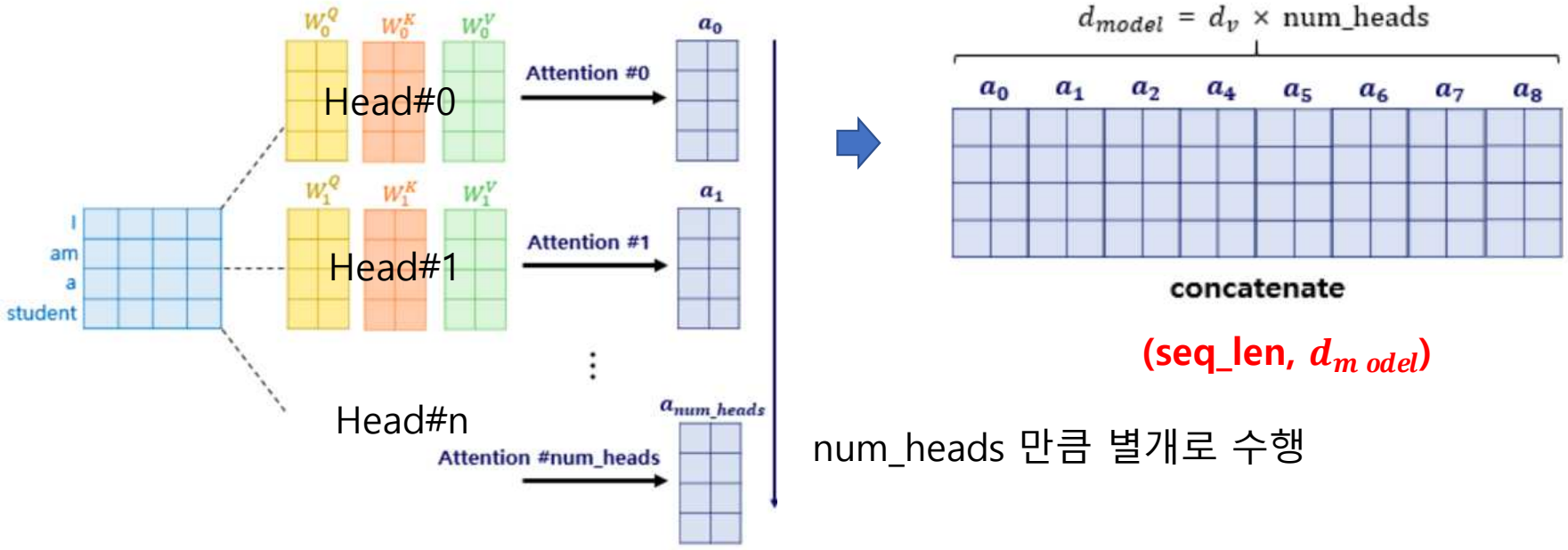
Normalization - 3단계

가중합 - 4단계

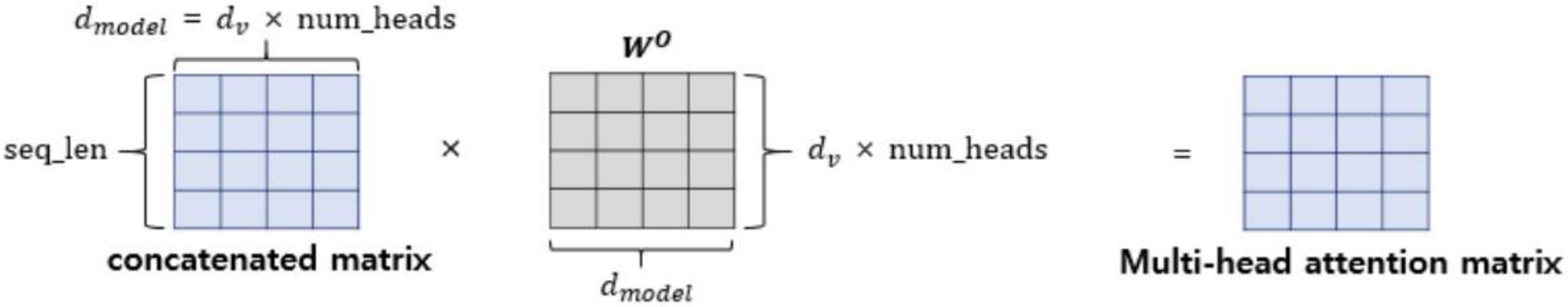
Attention Value
: 단어에 대한 Context Vector

# Multi-Head Attention

- 여러 개의 attention 을 병렬로 사용한 후 Attention Head 를 연결
  → 다른 시각으로 단어 간의 상관 관계 파악
- 각각이 random 하게 초기화 되므로 training 후 다른 subspace 표시



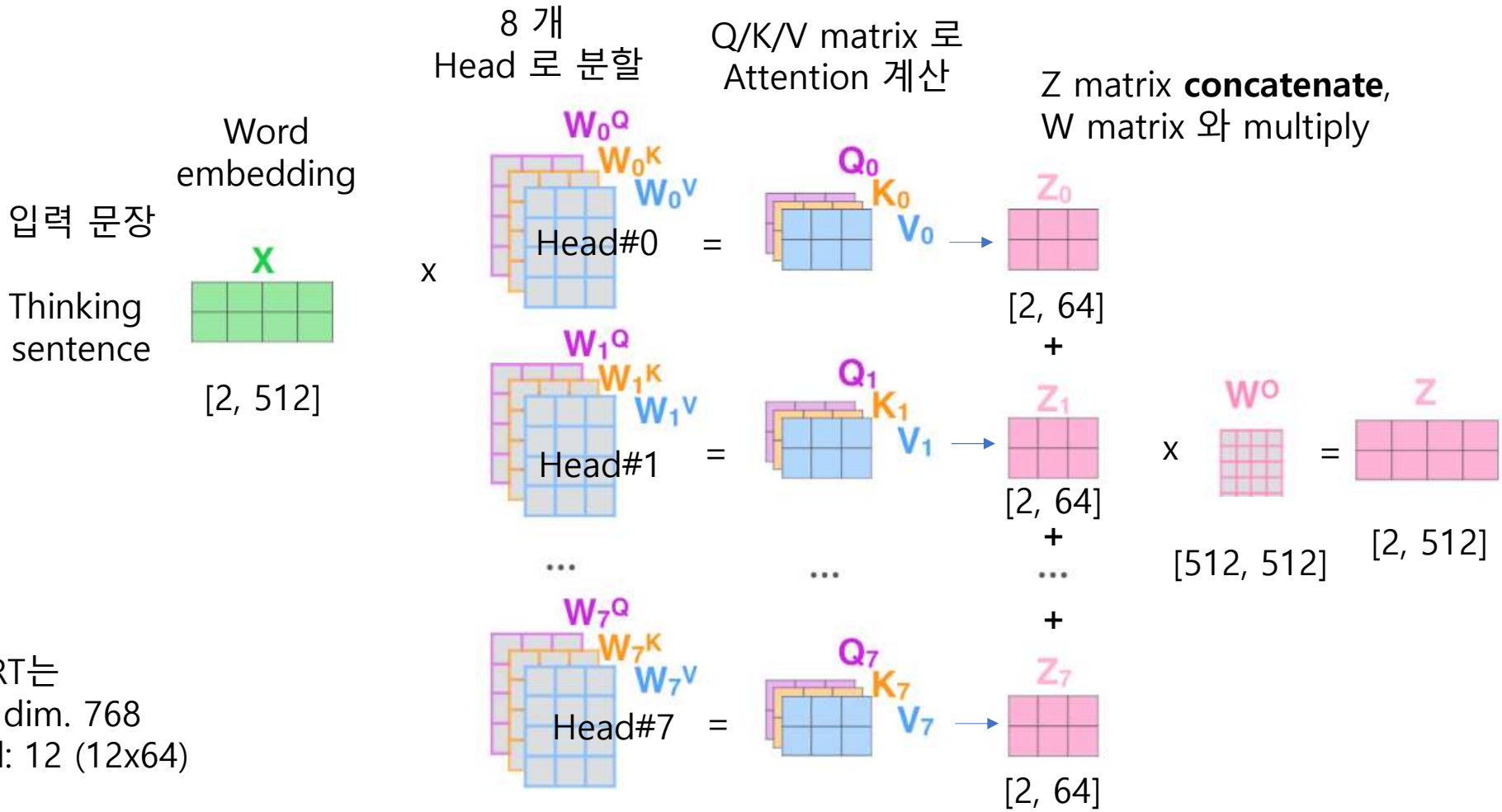
# Multi-Head Attention output matrix




**(seq_len, $d_{m\ odel}$)** **($d_{m\ odel}$, $d_{m\ odel}$)** **(seq_len, $d_{m\ odel}$)**

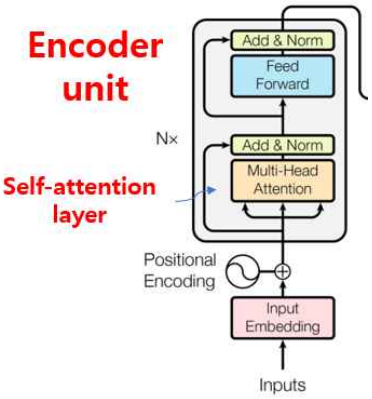
- Multi-Head Attention 의 크기는 Encoder 의 최초 입력의 크기와 동일
- Transformer 는 Encoder 를 6 개 쌓은 형태이므로, 입력의 크기가 출력에서 유지되어야 다음 Encoder 의 입력으로 사용 가능

# All together




* BERT는
EMB dim. 768
Head: 12 (12x64)

# Position-wise Feed Forward NN

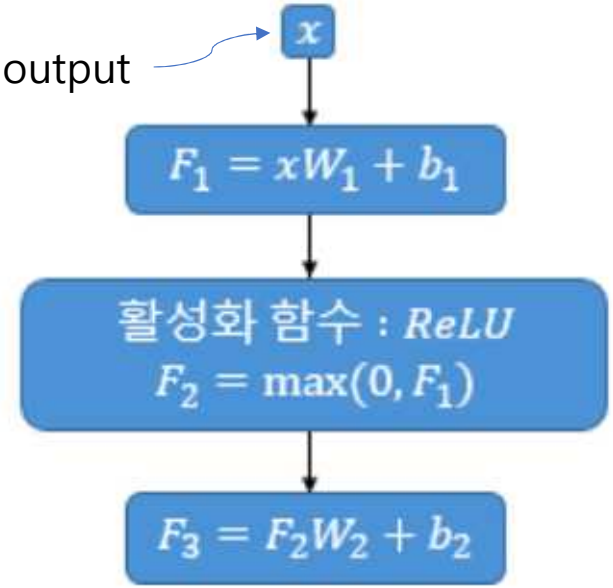


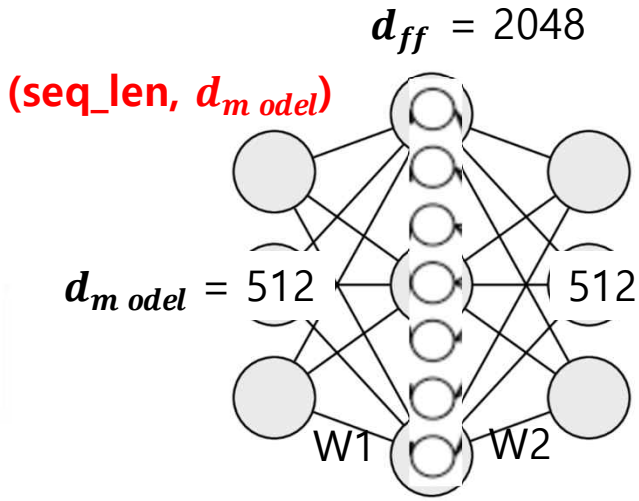

- Encoder 와 Decoder 의 각각의 layer 에서 보유
- Position 별 (단어별)로 별도로 적용하므로 position-wise

$$FFNN(x) = MAX(0, xW_1 + b_1)W_2 + b_2$$

- Fully-Connected FFNN

Multi-Head Attention 의 output




W1 : $(d_{model}, d_{ff})$ $(512, 2048)$

W2 : $(d_{ff}, d_{model})$ $(2048, 512)$

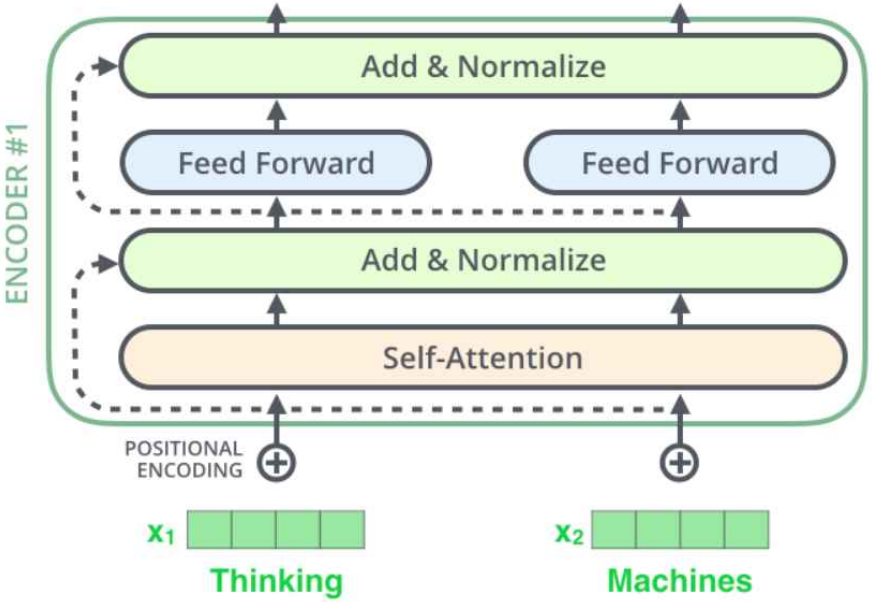
참고) BERT

$d_{model}$ = 768

$d_{ff}$ = 3072 (768x4)

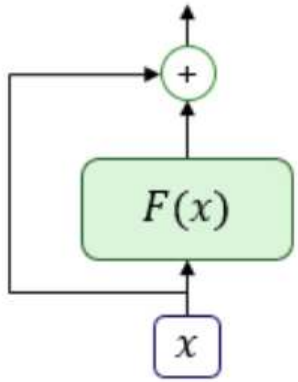
# 잔차 연결 및 정규화
# (Residual Connection and Layer Normalization)

- Add & Norm – 트랜스포머는 서브층의 입력과 출력이 동일한 차원을 유지하므로 잔차 연결 가능 ➔ Vanishing Gradient 해결



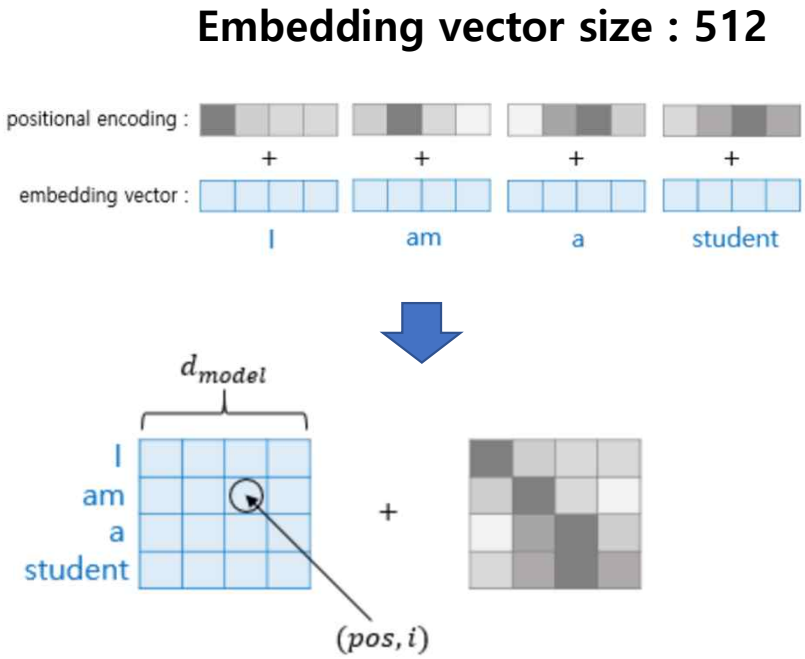

잔차 연결

$$H(x) = x + F(x)$$



# Positional Encoding

- 위치 정보 특성 해결
- Embedding vector 에 positional encoding 값 추가




- $PE_{(pos,2i)} = \sin(pos\ /10000^{2i/d_{model}})$ → 짝수 위치
- $PE_{(pos,2i+1)} = \cos(pos\ /10000^{2i/d_{model}})$ → 홀수 위치

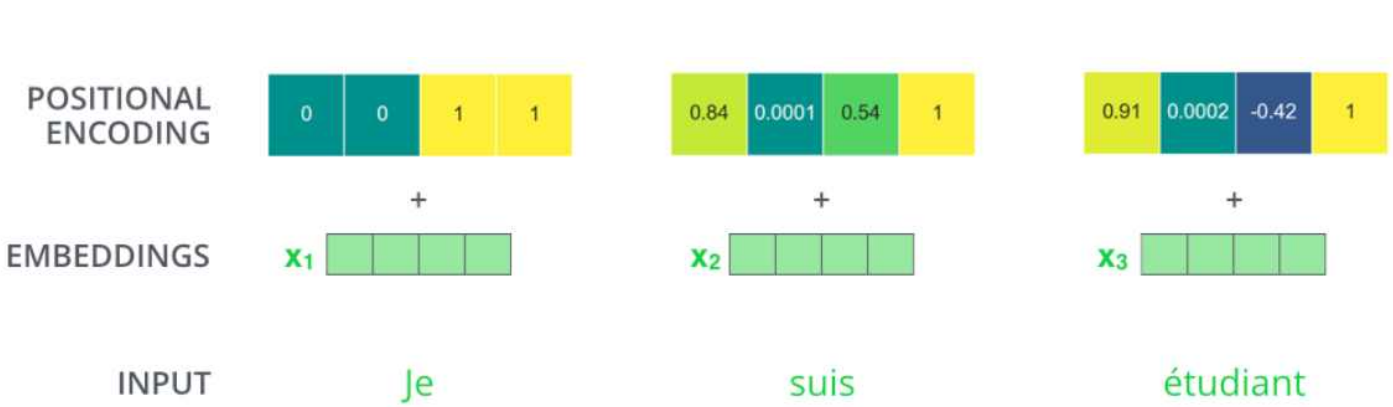
pos – 입력문장 내의 embedding vector 의 위치

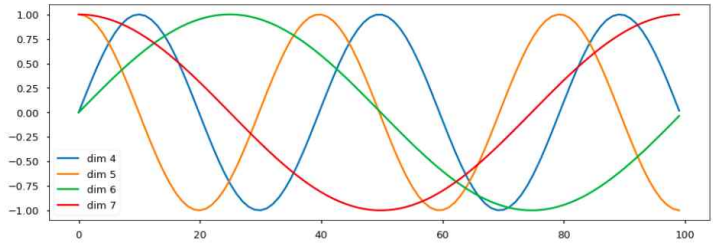
i – embedding vector 내의 차원의 index

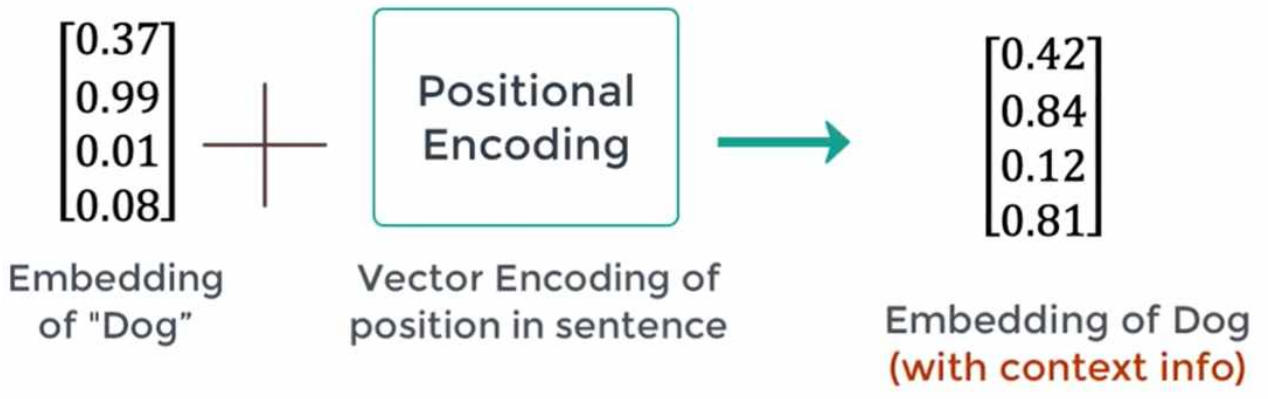
- 같은 단어라도 문장 내의 위치에 따라서 Transformer의 입력으로 들어가는 Embedding vector 값이 달라진다 → 순서의 정보가 포함된 Embedding Vector
- 정현파는 일정한 cycle 로 값이 규칙적으로 반복되므로, 신경망이 상대적 위치 값을 쉽게 학습

[유튜브 설명 동영상]

# Positional Encoding example




참고) BERT
Learnable Positional Embedding 사용

# Transformer 의 주요 Hyper-Parameter

- $d_{m\ odel} = 512$
  - encode 와 decoder 의 정해진 입출력 크기
  - Embedding vector 의 크기
  - 모든 encoder, decoder 층에 공통
- num_layers = 6
  - Encoder 와 decoder 의 전체 층수
- num_heads = 8
  - 병렬처리 multi-head 갯수
- $d_{ff} = 2048$
  - 트랜스포머 내부 feed forward 신경망의 은닉층 크기

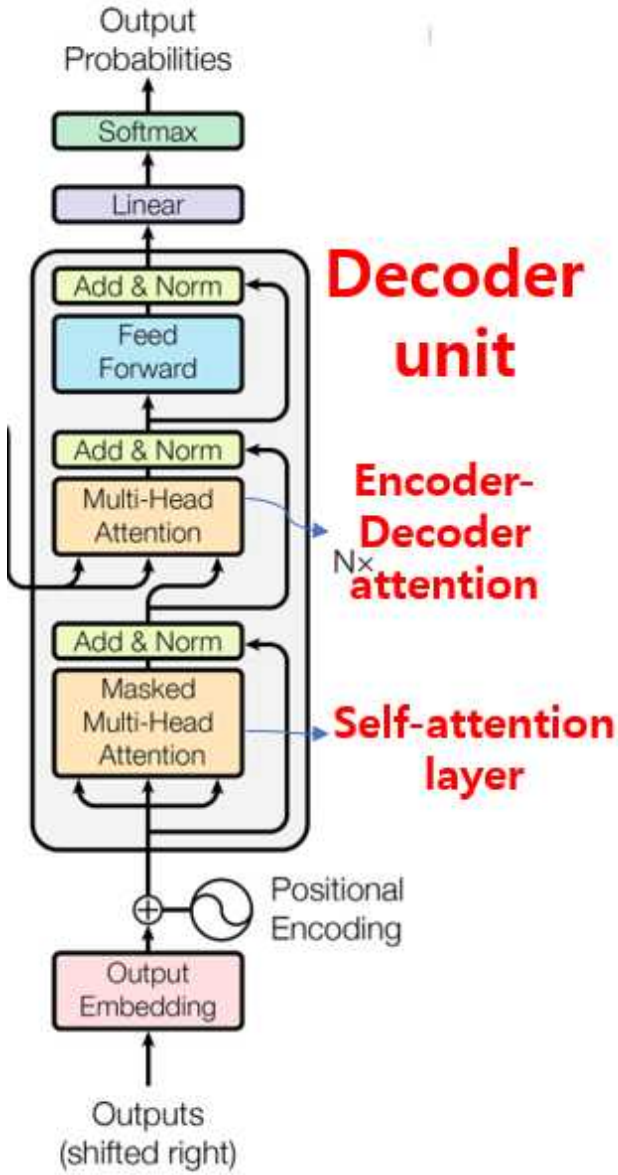
# Decoder

- **Top Encoder 의 output** 이 **Decoder 의** attention vector **Key, Value 로 transform** 되어 각 decoder 의 "encoder-decoder attention" layer 에서 사용
  → Input sequence 의 적절한 위치를 focus 하도록 함

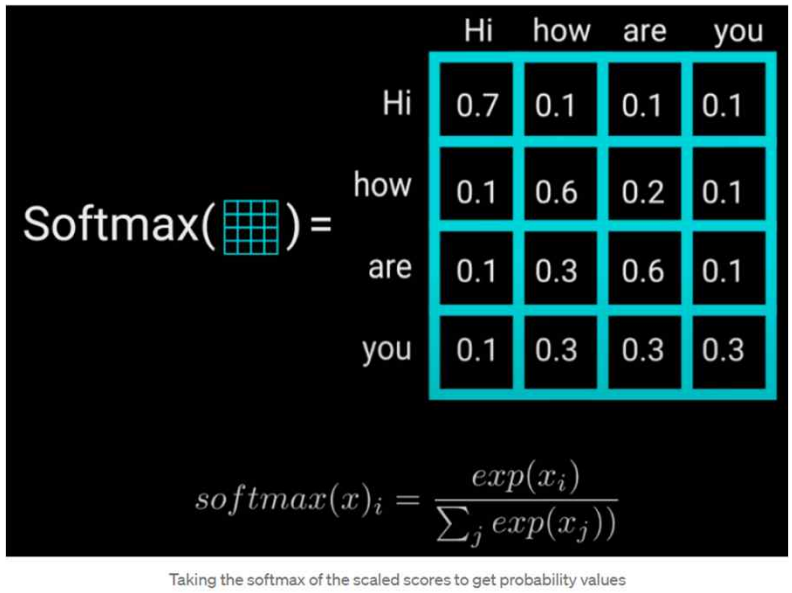
- **Query** : training - ground truth label 사용
  inference – previous predictions 사용

- Decoding result 는 encoder 경우처럼 위로 전달
  → 각 step 의 output 은 next time step 에서 bottom decoder 의 input 으로 사용

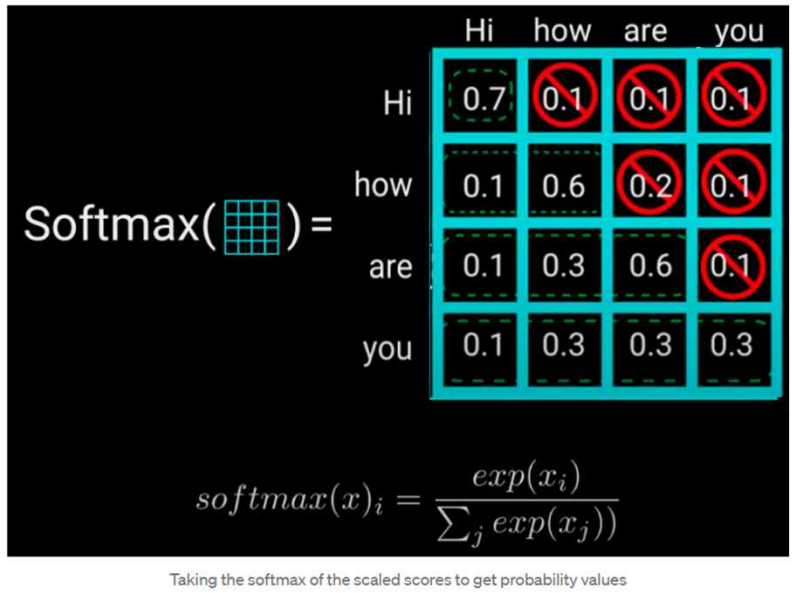
- Decoder input 에 positional encoding 을 추가해 줌



- Decoder 의 self-attention layer 는 output sequence 의 self-attention 계산을 위한 softmax step 전에는 **future position 을 masking** (-inf 로 setting) 하여 이전 위치만 attend 할 수 있도록 한다.

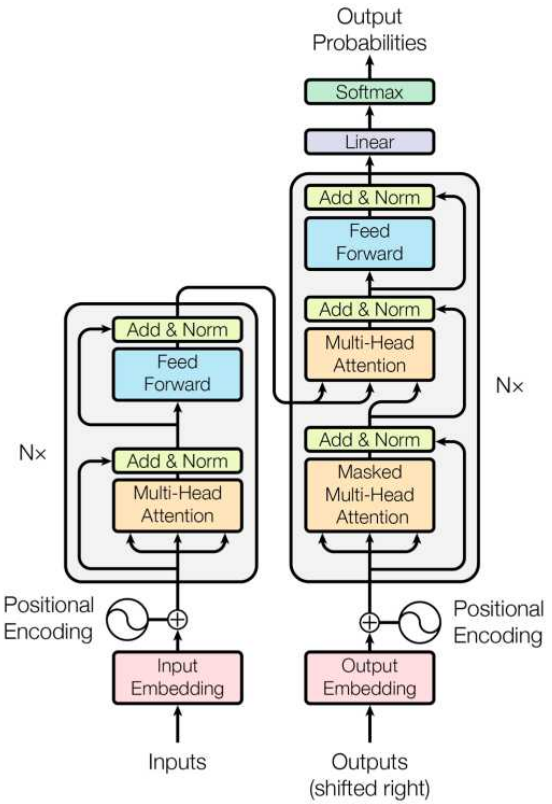


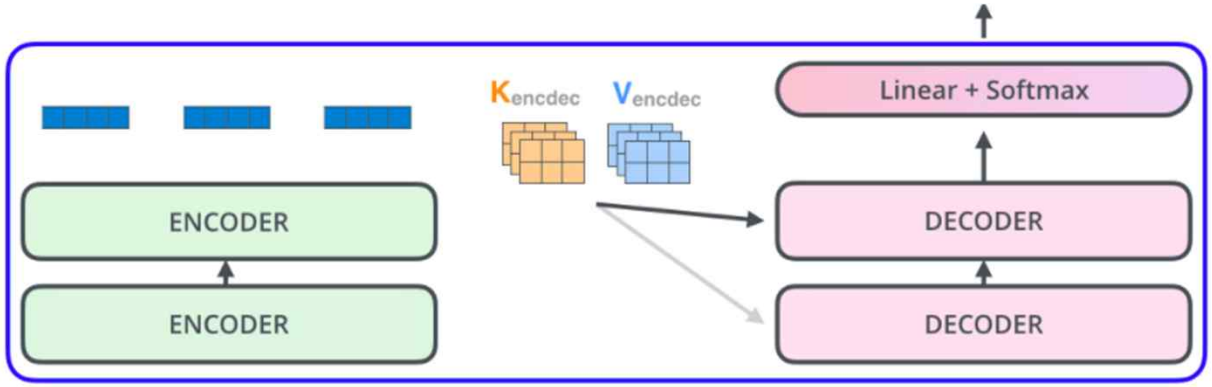

Encoder 의 attention score 계산




Decoder 의 attention score 계산

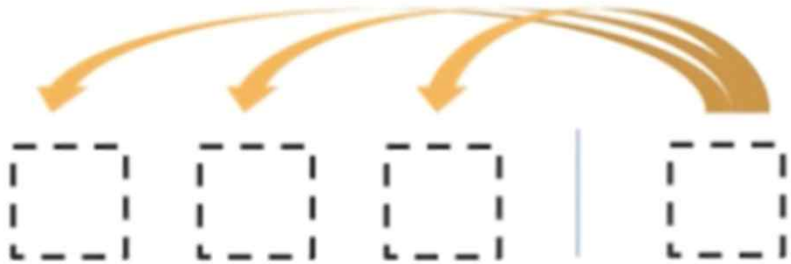
- Encoder-Decoder attention 은 Query 가 Decoder 의 아래쪽 layer 에서 생성되는 것 외에는 Encoder 의 multi-headed attention 과 동일하게 동작



# Transformer 의 3 가지 Attention 정리

- key, value from encoder



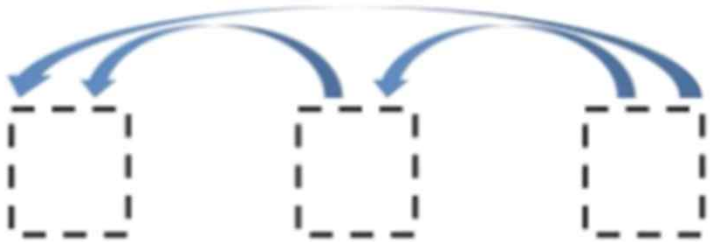
query

Encoder-Decoder Attention

- query, key, value
- Multi-head attention

Encoder Self-Attention

- query, key, value
- Masked Multi-head attention

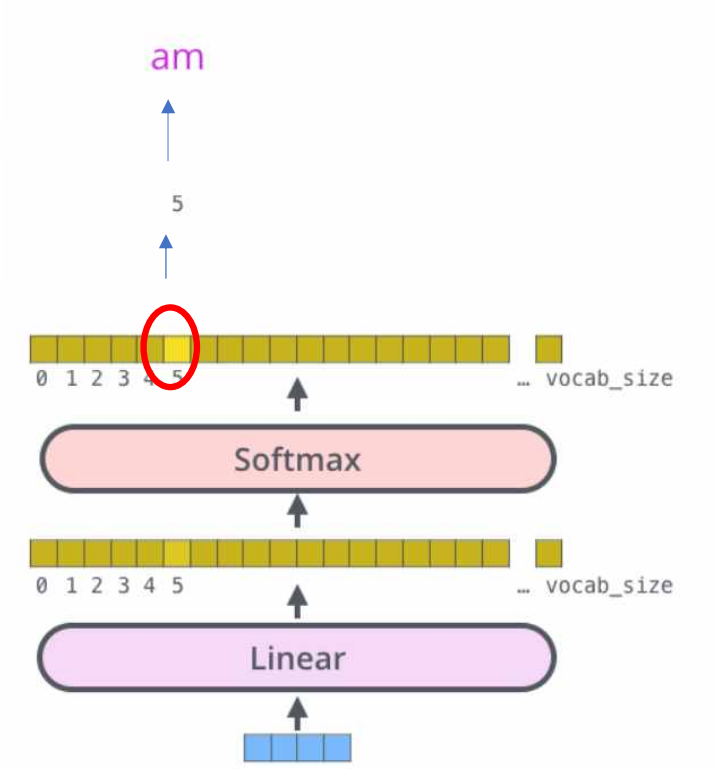


MaskedDecoder Self-Attention

# Final Linear and Softmax Layer

Vocabulary 의
해당 index 의 단어

가장 큰 값의 index 선택
(argmax)




**Label Smoothing**

– overfitting 방지를 위해 1 보다는 작고,
0 보다는 크도록 one-hot encoding 조정

Ex) Thank you
– 감사합니다. 고마워. 모두 정답

# 실습: 130-Transformer model 이해

- From Google Tutorial
  (https://www.tensorflow.org/tutorials/text/transformer)

- Portugal – English translation
- Text generation 및 Attention 이해 필수
- Encoder-Decoder model
- Transformer model 의 핵심은 Self-Attention

# Transfer Learning of NLP

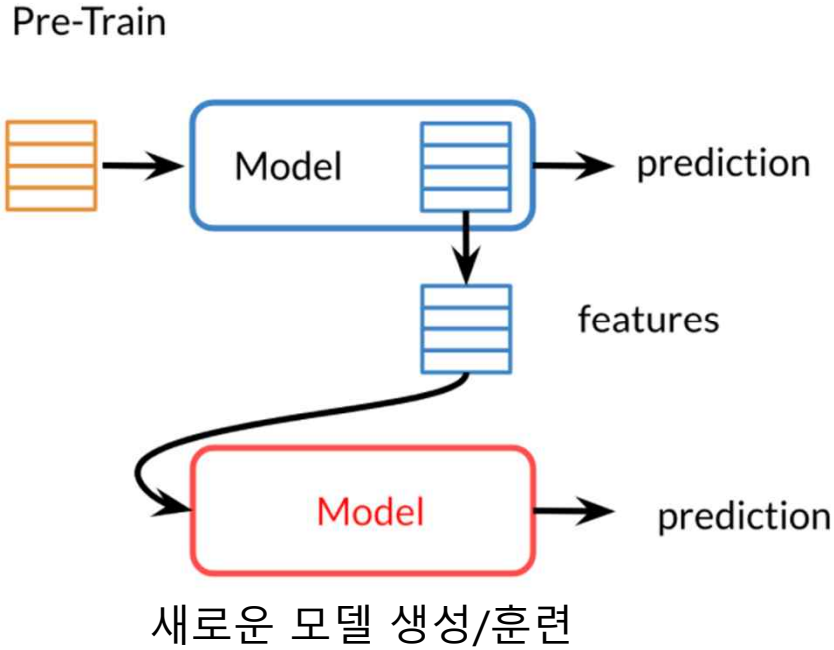
# Transfer Learning

- Training Time 절감
  - GPT-3 와 같은 최첨단 model 은 train 에 약 2 개월 소요
- 예측 성능 향상
- Small dataset 으로 첨단 performance 구현

# 전이 학습 (Transfer Learning) 방법

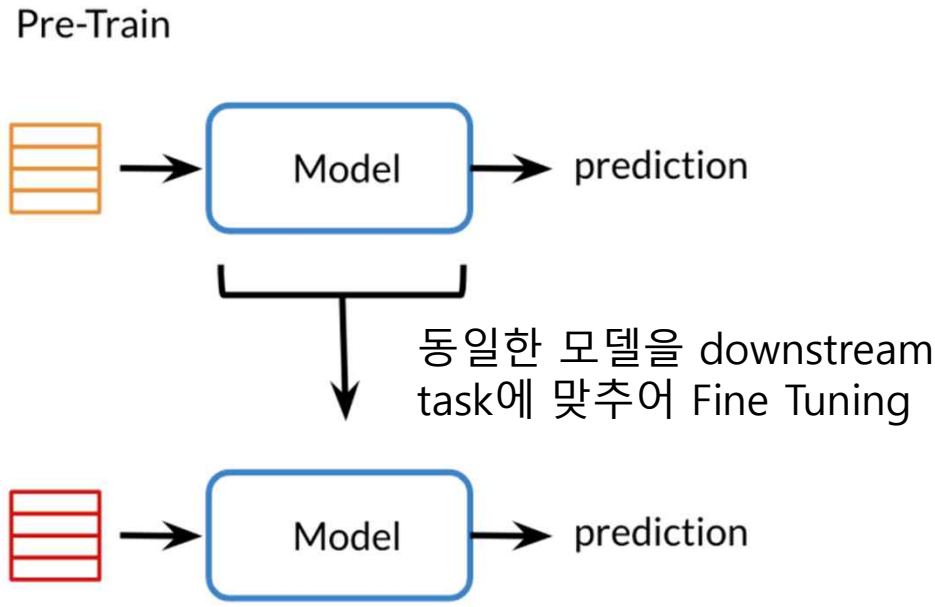
**Feature-based Transfer Learning**

pre-trained model의 출력 feature를 new model 의 입력으로 사용

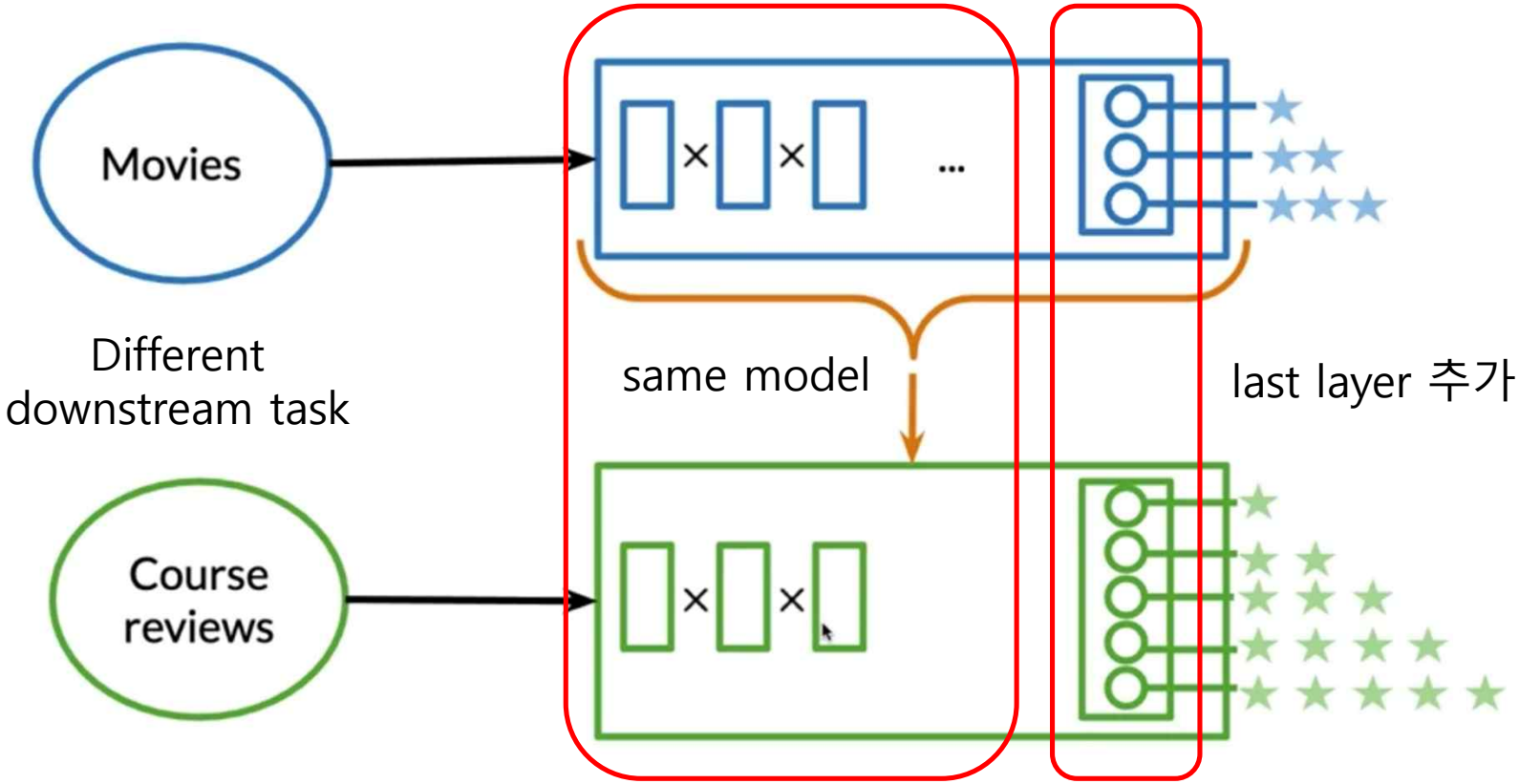



**Fine-Tuning**

pre-trained weight 를 초기값으로 사용하여 동일 model fine-tuning



# Fine-tuning – last layer 추가



# Advantage of Fine-Tuning

- BERT에는 이미 우리 언어의 많은 것이 인코딩 되어 있다. 따라서,
  - 더 빠른 개발
  - 필요한 데이터 감소
  - 더 나은 결과를 얻을 수 있다.

- 단점
  - Large size
  - Slow fine-tuning
  - Slow inferencing
  - 특정 domain 에 특화된 단어를 모름

| Layer | Weights |
|---|---|
| Embedding Layer | ~24M |
| Transformers (x12) | ~7M ea. x 12 = ~85M |
| **TOTAL** | **~109M** |

(417 MB)

# BERT

(Bidirectional Encoder Representations from Transformers)

# BERT

- 2018. Nov. 발표 (https://github.com/google-research/bert)
  - 104 개 multi-language model 동시 발표
  - ALBERT, RoBERTA, TinyBERT, DistilBERT, and SpanBERT 등 다양한 변종
- 질의 응답 시스템, 감성분석등의 다양한 GLUE task 에서 SOTA 달성
- Pre-trained token embedding (WordPiece) 사용

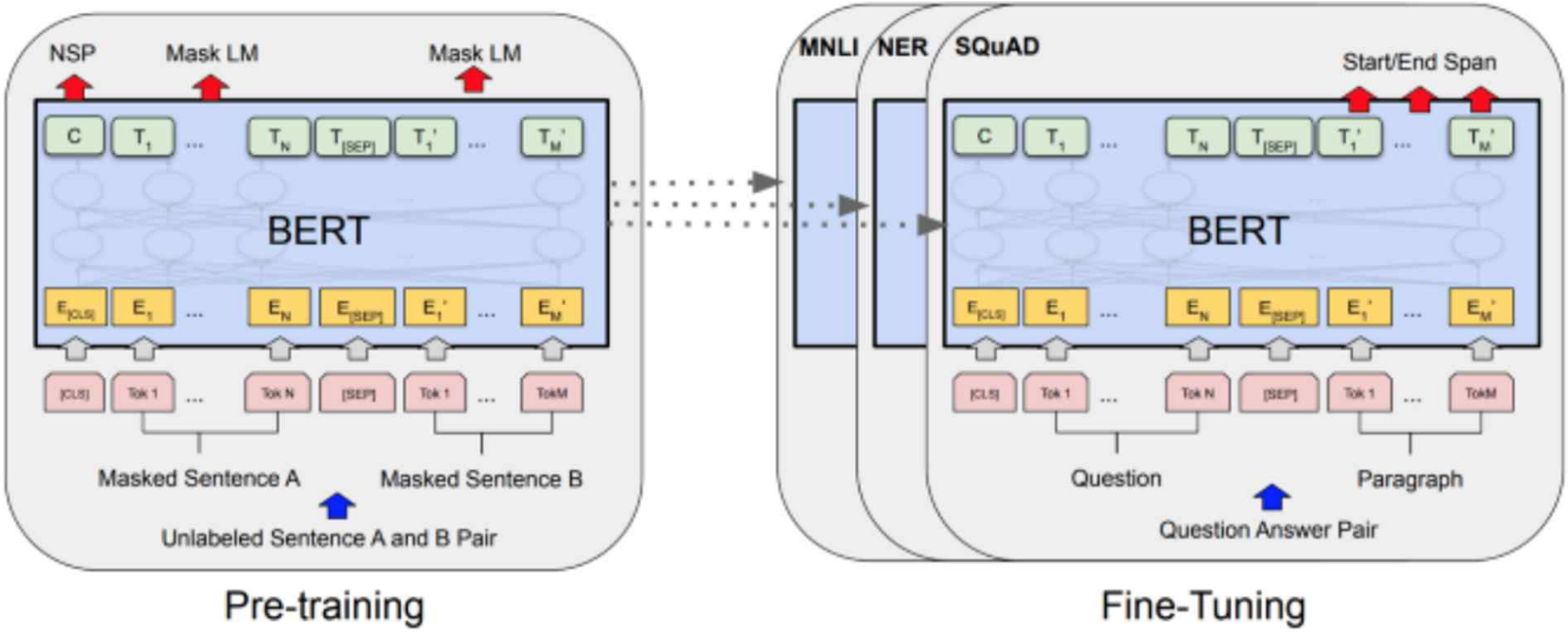
  → 30,000(영문 version) / 110,000(104개국어 version) subwords
- BERT-Base model : Transformer layer 12 개, Total parameters 110M
- BERT-Large model : Transformer layer 24 개, Total parameters 340M

# BERT 의 접근 방식

1) 범용 Solution 을 (Transformer 의 encoder 이용)
2) Scalable 한 형태로 구현하여
3) 많은 머신 리소스로 훈련하여 성능을 높인다.
    - Pre-training 에 사용한 data
        - BooksCorpus (800M 단어)
        - English Wikipedia (2500M 단어)
    - BERT-Base : 64 TPU 4 days 소요

- 11 개의 NLP 과제에 하나의 pre-trained model 을 공통적으로 사용. 단, 각각의 과제에 대해 약간의 fine-tuning 적용
  - 발표 당시 SQuAD(Standford Question Answering Dataset) 에서 human level 능가
- NLP 를 위한 AI solution 개발은 coding 능력을 갖춘 모든 사람에게 개방

# BERT 전략




- No human labeling (비지도 학습)
- 대규모 corpus 사용
- 대규모 resource 사용

- 다양한 downstream NLP task 에 적용
  - MNLI(Multi-Genre Natural Language Inference)
  - Named Entity Recongnition
  - SQuAD

# BERT 적용 범위

- 모든 NLP 문제에 적용되지는 않는다

**CAN DO**

- Classification
- NER (Named Entity Recognition)
- POS Tagging
- Question Answering

**CANNOT DO**

- Language Model (next word 예측)
- Text Generation
- Translation

# GLUE Benchmark

- GLUE (General Language Understanding Evaluation) 는 한개의 자연어 처리 모델에 대해 여러 태스크들을 훈련시키고, 그 성능을 평가 및 비교 분석하기 위한 데이터셋들로 구성
- 모델들의 **자연어 이해 능력**을 평가하기 위해 다양하고 해결하기 어려운 9개의 Task Dataset 으로 구성
- BERT와 같은 전이학습 모델들을 평가하기 위한 필수적인 벤치마크

| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.9 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 88.1 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.2 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.1 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **91.1** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **81.9** |

# GLUE Task 구성 – 9 개

- CoLA (The Corpus of Linguistic Acceptability)
  - 문장이 문법적으로 맞는지 분류 (이진분류)
- SST-2 (The Stanford Sentiment Treebank)
  - 영화평 감성 분류 (positive, negative, neutral)
- MRPC (Microsoft Research Paraphrase Corpus)
  - 문장B 가 문장A 의 다른 표현 (paraphrase) 인지 여부 (이진분류)
- STS-B (The Semantic Textual Similarity Benchmark)
  - 문장 A 와 B 는 얼마나 유사한가 ?
- QQP (Quora Question Pairs)
  - 두개의 질문이 서로 유사한가 ? (이진분류)

- MNLI (Multi-Genre Natural Language Inference)
  - 문장 B 가 문장 A 에 이어지는 문장인지, 반대되는 문장인지, 무관한 문장인지 여부
  - 문장 A (Hypothesis) [sep] 문장 B (Premise)

- QNLI (Question Natural Language Inference)
  - 문장 B가 문장 A의 질문에 대한 답을 포함하는지 여부 (이진분류)

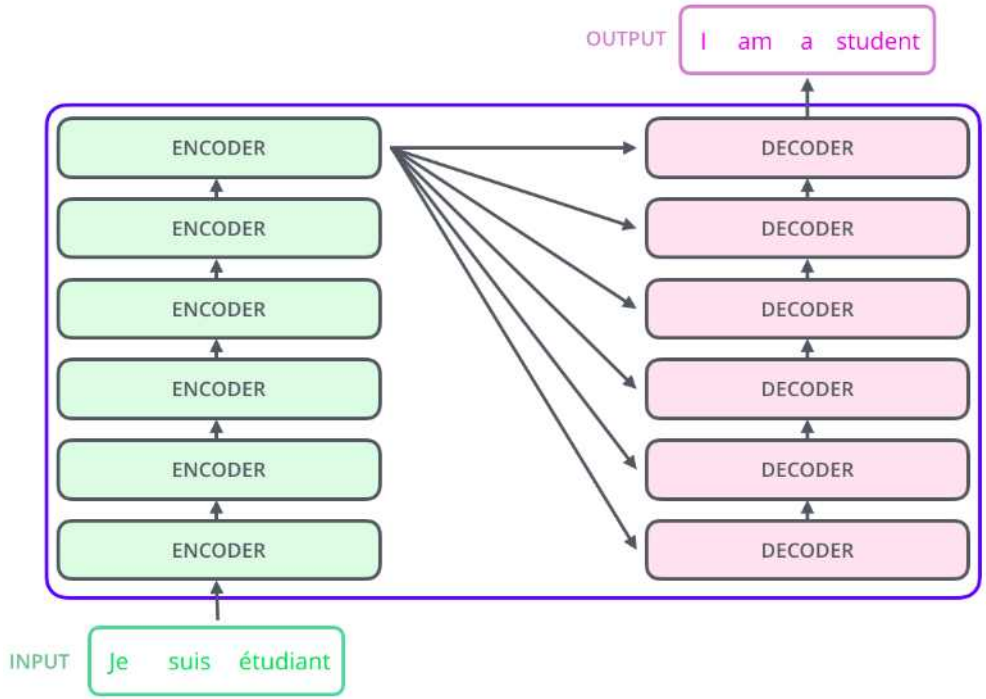
- RTE (Recognizing Textual Entailment)
  - MNLI 와 유사하나, 상대적으로 훨씬 적은 데이터셋 (이진분류)

- WNLI (Winograd NLI)
  - 문장 B가 문장 A의 애매한 대명사를 정확한 명사로 대체하였는가 ? (이진분류)

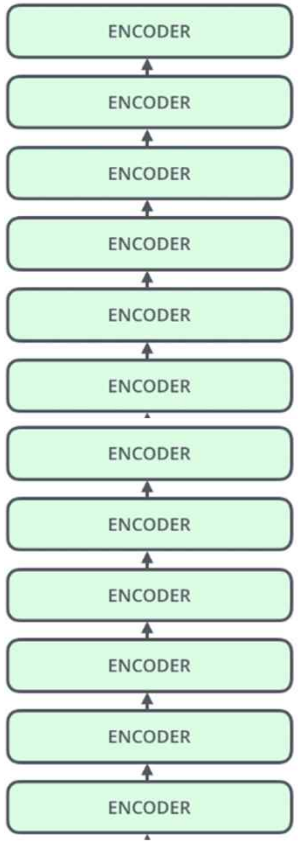
- 기타 non-GLUE : SQuAD (Stanford NLP Group, Wikpedia 질의 응답)

# BERT 구성

**Transformer**
**6 encoders + 6 decoders**




**BERT**
**12/24 encoders**

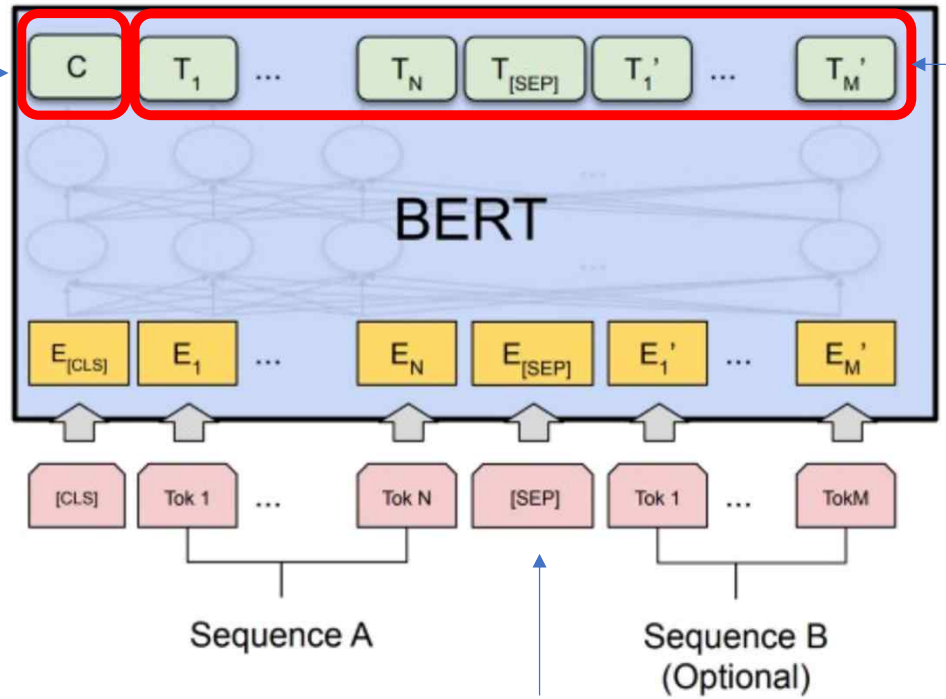



**MLM, Next Sentence Prediction**

# How to train BERT ?

Output token 갯수는
input 과 동일

C ([CLS]) 는 모든 NLP
분류 문제에
prediction 으로 사용




non-classification task 에서
사용

모든 input 은 [CLS]
token 으로 시작

Input 이 두개의
sequence 이면 중간
에 [SEP] token 삽입

**1) Masked Language Modeling**

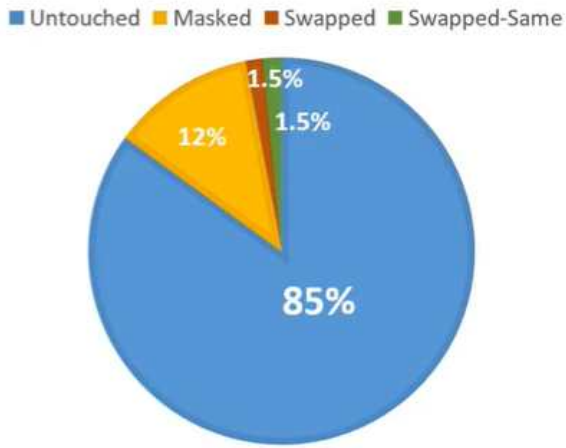
**2가지 Task**

**2) Next Sentence Prediction**

# Train 1 : Masked Language Modeling (MLM)

- Input 에서 random 하게 몇 개의 token 을 masking
  - **Input:** the man went to the [MASK] to buy a [MASK] of milk
    **Label:** store gallon

- Left, right context 만을 가지고 masked word 의 원래 단어를 예측
  - 전체 학습 데이터 token 의 15% 를 masking.
  - 이중 80% 는 [MASK] token 으로 replace
  - 10% 는 random word 로 replace
  - 10% 는 unchanged (tagging 만 바뀐 것으로 표시)



# Train 2 : Next Sentence Prediction

- Training data 로 sentence pair 사용
- 주요 목적은 output 의 C([CLS]) 를 train 시키는 것
- 예를 들어 100,000 개의 sentence 가 있고 BERT language model 을 pre-train 시키려면 50,000 개 pair 의 training data 존재
  - 50% 의 pair 는 second sentence 가 실제 다음 sentence (corpus 에서 연속된 문장)
  - 나머지 50% pair 는 second sentence 를 corpus 에서 random 선택 (사람의 개입 없음)
  - 첫째 경우의 label 은 'IsNext' (첫번 문장과 다음 문장이 관련 있음) 이고 둘째 경우는 'NotNext' (관련 없음)이다.

- 이어지는 문장이 앞선 문장과 연결되었는지 여부는 다음과 같이 train
  - 전체 input sequence 를 Transformer model 에 입력
  - [CLS] token 의 output 을 simple classification layer 를 이용하여 2×1 shaped vector 로 변환
  - Softmax 를 이용하여 IsNextSequence 예측
- BERT model train 시, **Masked LM 과 Next Sentence Prediction 은 동시에 train** 되고 goal 은 **combined loss function 을 minimize** 하는 것

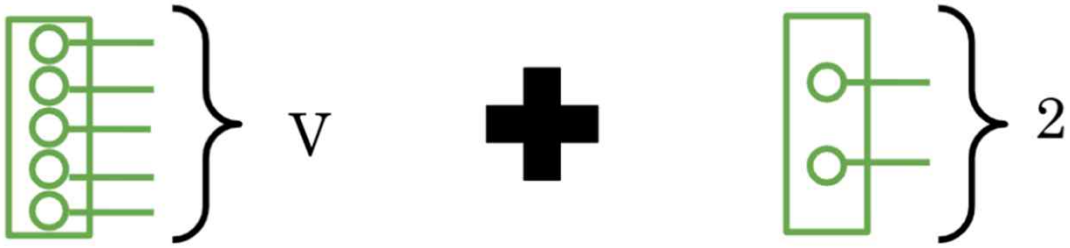
# BERT 의 목적 함수

Objective 1:
Multi-Mask LM

Loss: Cross Entropy Loss

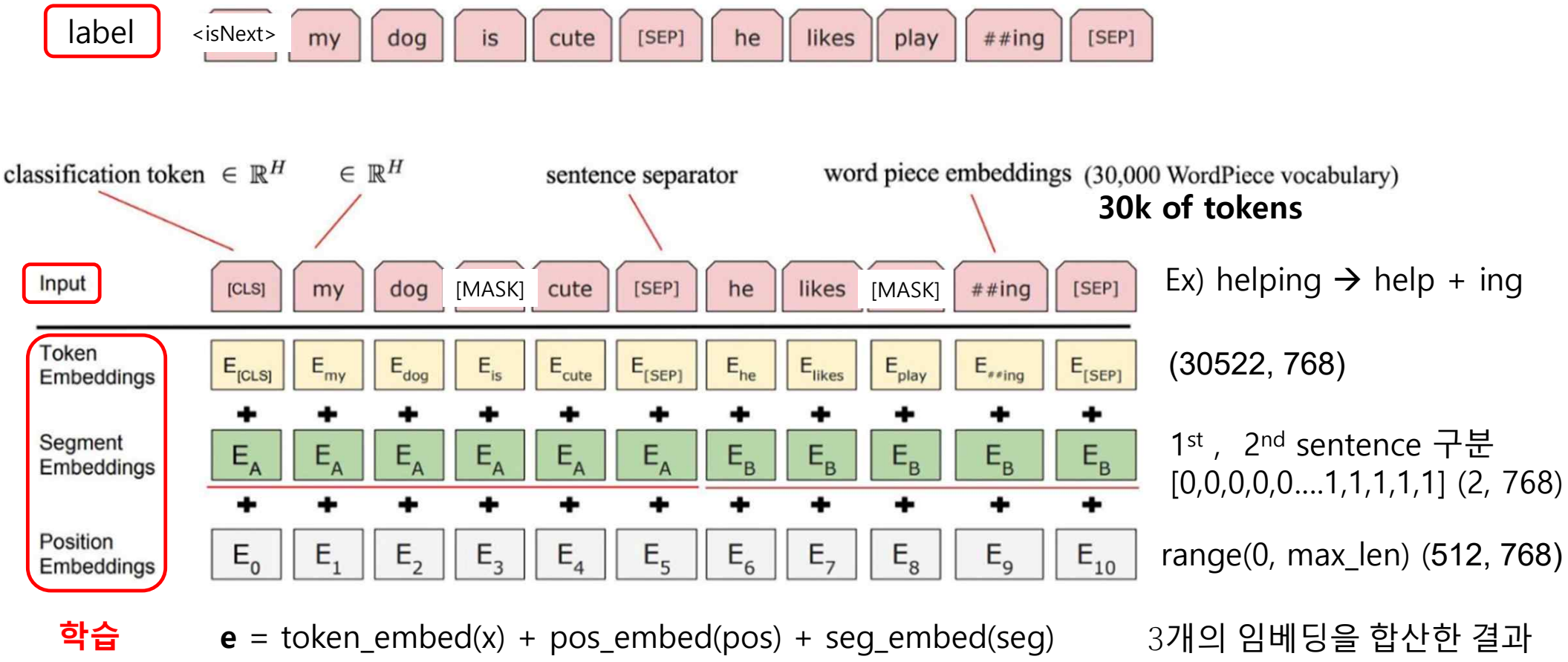
Objective 2:
Next Sentence Prediction

Loss: Binary Loss




두 가지 목적함수를 합산

# BERT 의 Input Embeddings – All Together

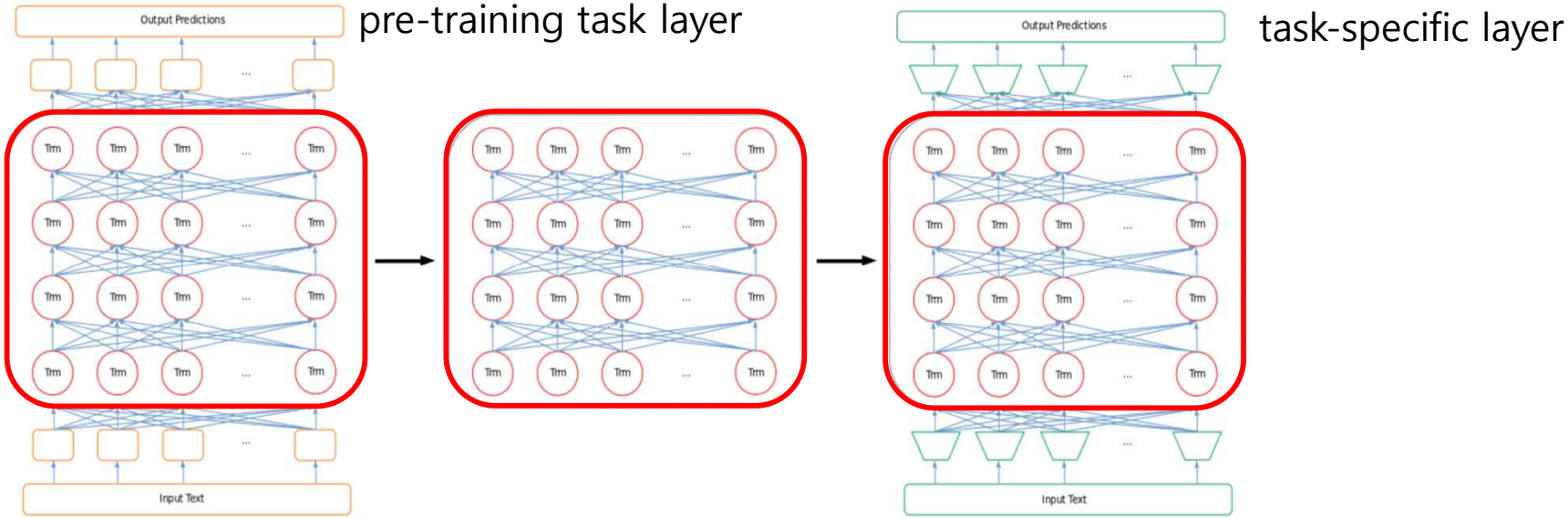



**학습** **e** = token_embed(x) + pos_embed(pos) + seg_embed(seg) 3개의 임베딩을 합산한 결과

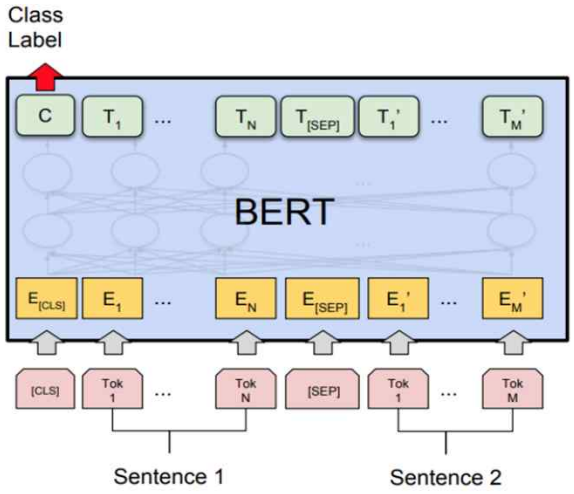
# Fine-tuning process of BERT

- 위 과정을 거쳐 변형된 모델을 n번의 epoch 동안 재학습(Fine-Tuning)

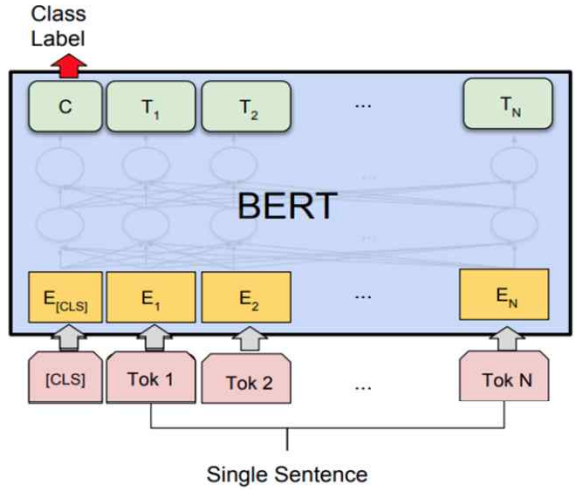
- 이때 사전학습 모델에서 차용한 모델 중간부는 자연어를 잘 '이해'하는 파라미터를 지니고 있기 때문에 우리가 풀고자 하는 문제 해결에 큰 도움이 됨



## Classification Task



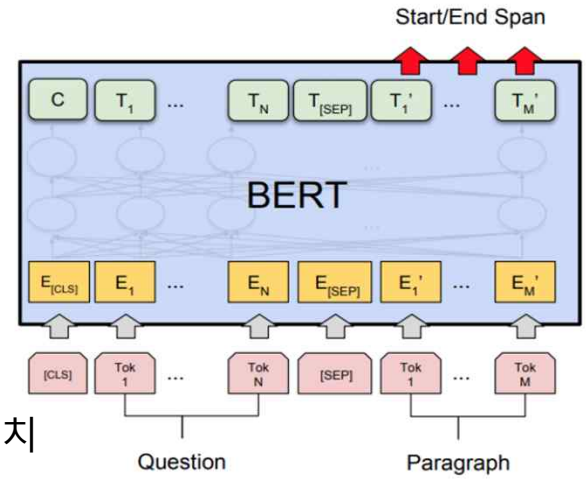

(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



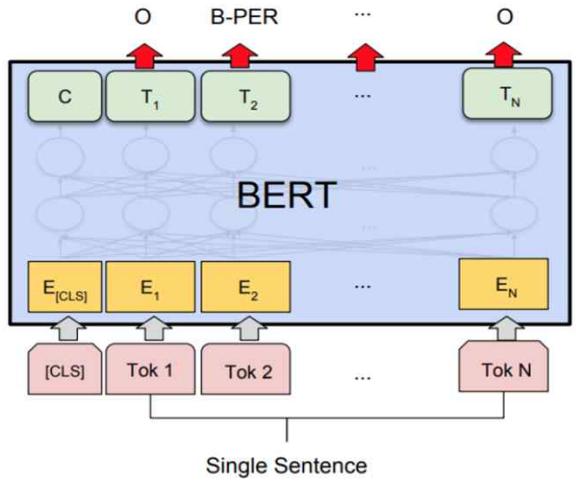

(b) Single Sentence Classification Tasks: SST-2, CoLA

## Non-Classification Task

Question – 질문
Paragraph – 본문
Start – 분문 중 정답 시작 위치
End – 본문 중 정답 끝 위치



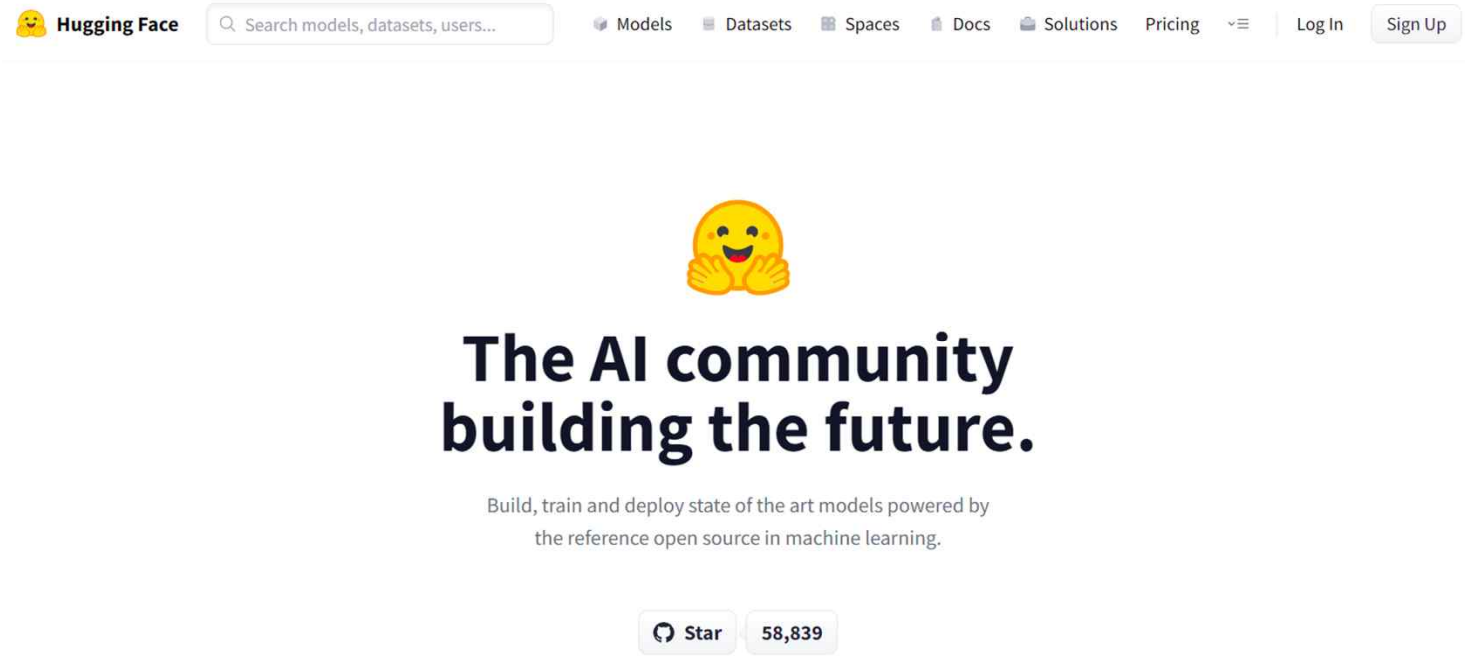

(c) Question Answering Tasks: SQuAD v1.1




(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

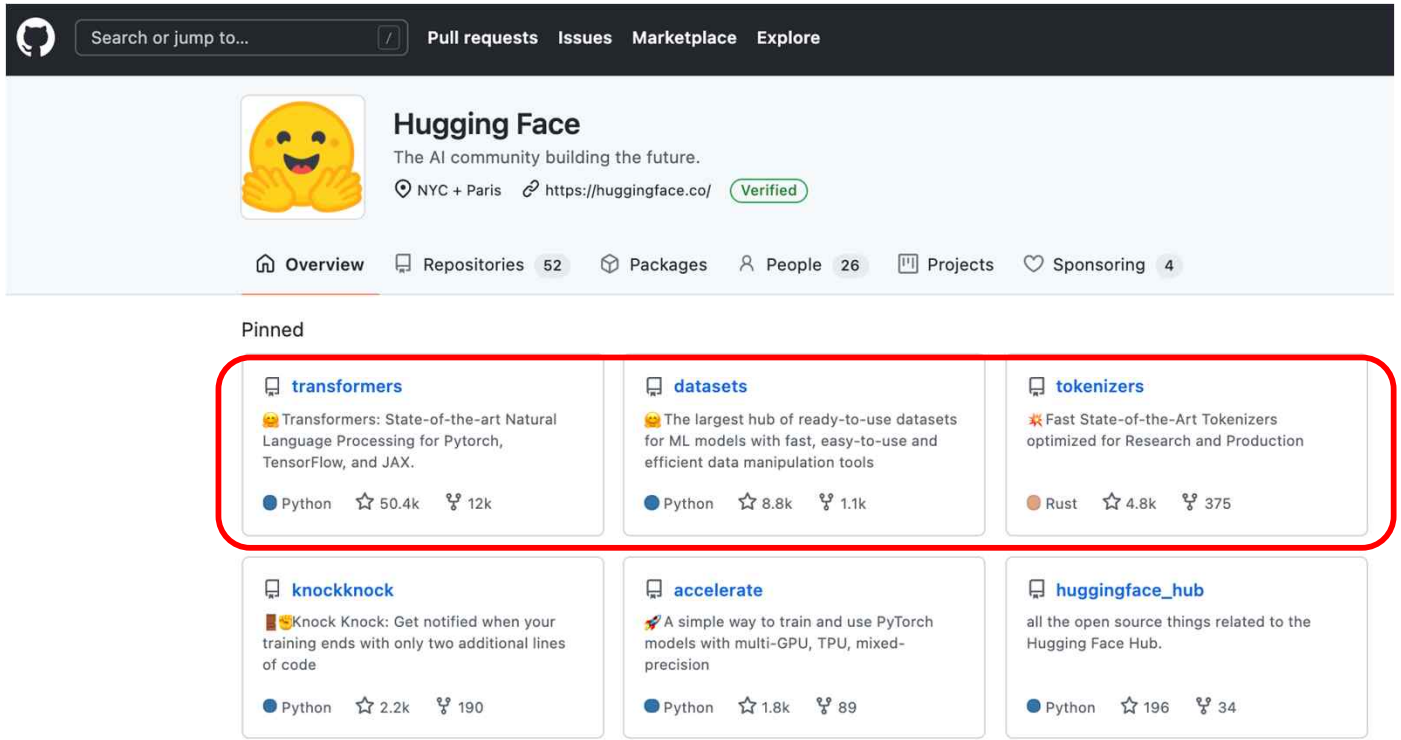
# HuggingFace transformers library (https://huggingface.co/)

- NLP 기술을 많은 이들이 편히 이용할 수 있도록 기술민주화를 하는 그룹



# Hugging Face (https://github.com/huggingface)

- NLP 와 관련된 다양한 패키지를 제공



# Hugging Face

- Transformers
  - Transformer 기반 (masked) language model 알고리즘 제공
  - 기 학습된 (pretrained) 모델 배포

- Tokenizers
  - transformers package 에서 사용할 수 있는 subword 토크나이저 학습
  - transformers 와 분리되어 다른 목적에도 이용할 수 있다.

- dataset → tokenizer → models 의 언어 모델 학습에 필요한 전 과정을 지원

# Hugging Face Trainer

- Transformers 라이브러리에서 제공하는 고수준 API로
- 사전 학습된 모델의 미세 조정(fine-tuning)과 평가를 간편하게 수행
- 복잡한 학습 루프를 직접 구현할 필요 없이, 모델 학습에 필요한 주요 구성 요소들을 설정하여 효율적으로 모델 훈련
- 주요 구성 요소
  1. 모델 (model): 사전 학습된 모델 객체 (예: AutoModelForSequenceClassification)
  2. 학습 인자 (TrainingArguments): 출력 디렉토리, 에폭 수, 배치 크기, 평가 전략 등 정의
  3. 데이터셋 (train_dataset, eval_dataset): 학습 및 평가에 사용할 데이터셋
  4. 토크나이저 (tokenizer): 텍스트 데이터를 모델 입력 형식으로 변환하는 도구
  5. 데이터 콜레이터 (data_collator): 배치 단위로 데이터를 처리하는 함수.
  6. 평가지표 함수 (compute_metrics): 모델의 성능을 평가하기 위한 함수

# 실습: 320-HuggingFace Sentiment Fine Tuning

- NAVER movie review dataset을 이용하여 BERT model을 fine tuning
- 다국어 version인 'bert-base-multilingual-cased' Tokenizer 를 이용한 dataset encoding
- Trainer 를 이용한 model fine tuning
- Fine Tuning 한 model 을 이용한 예측

# GPT-3

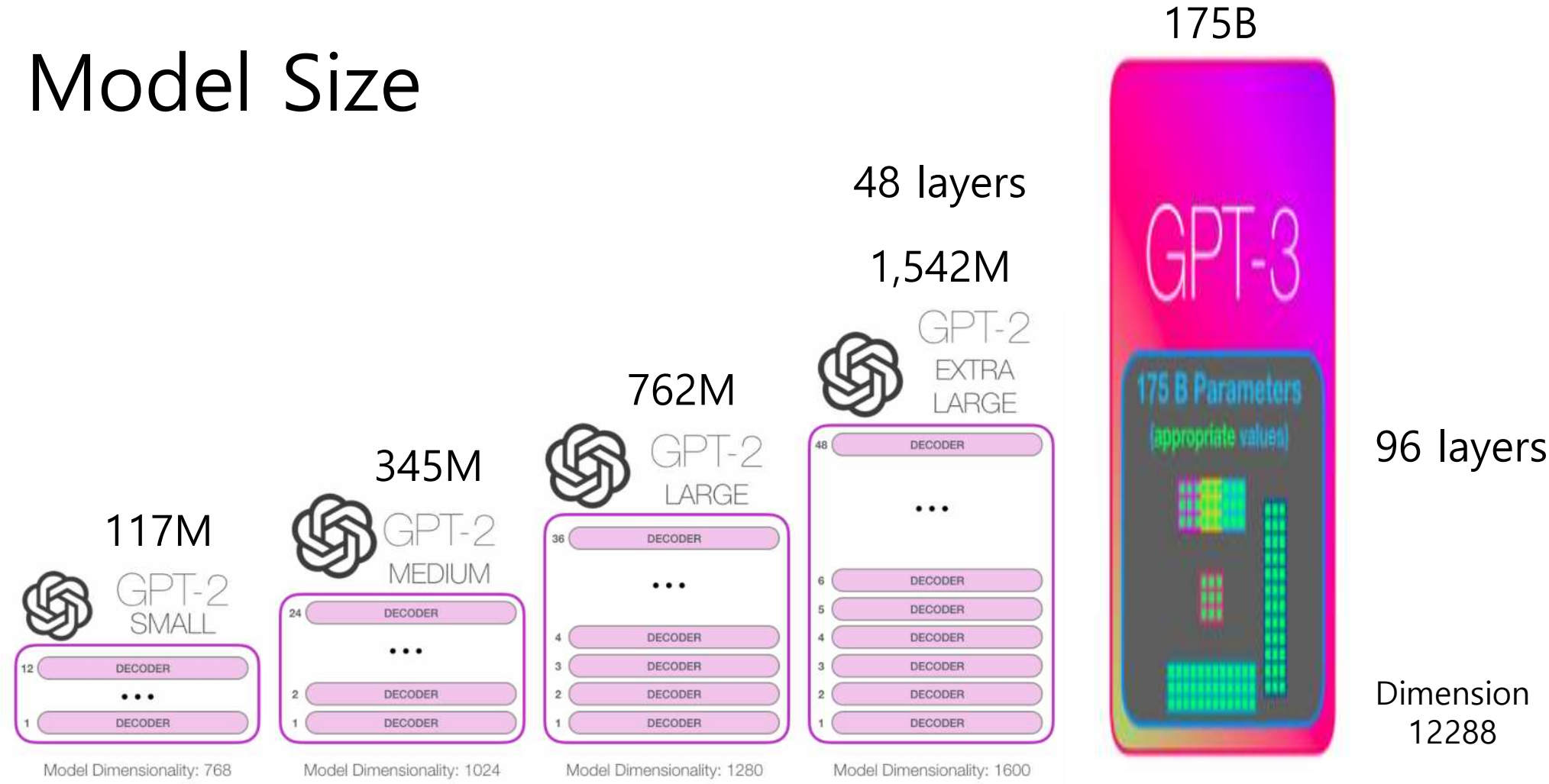
# What is GPT-3 ?

- Transformer의 decoder 부분을 가능한 높이 쌓은 구성(96 layers)

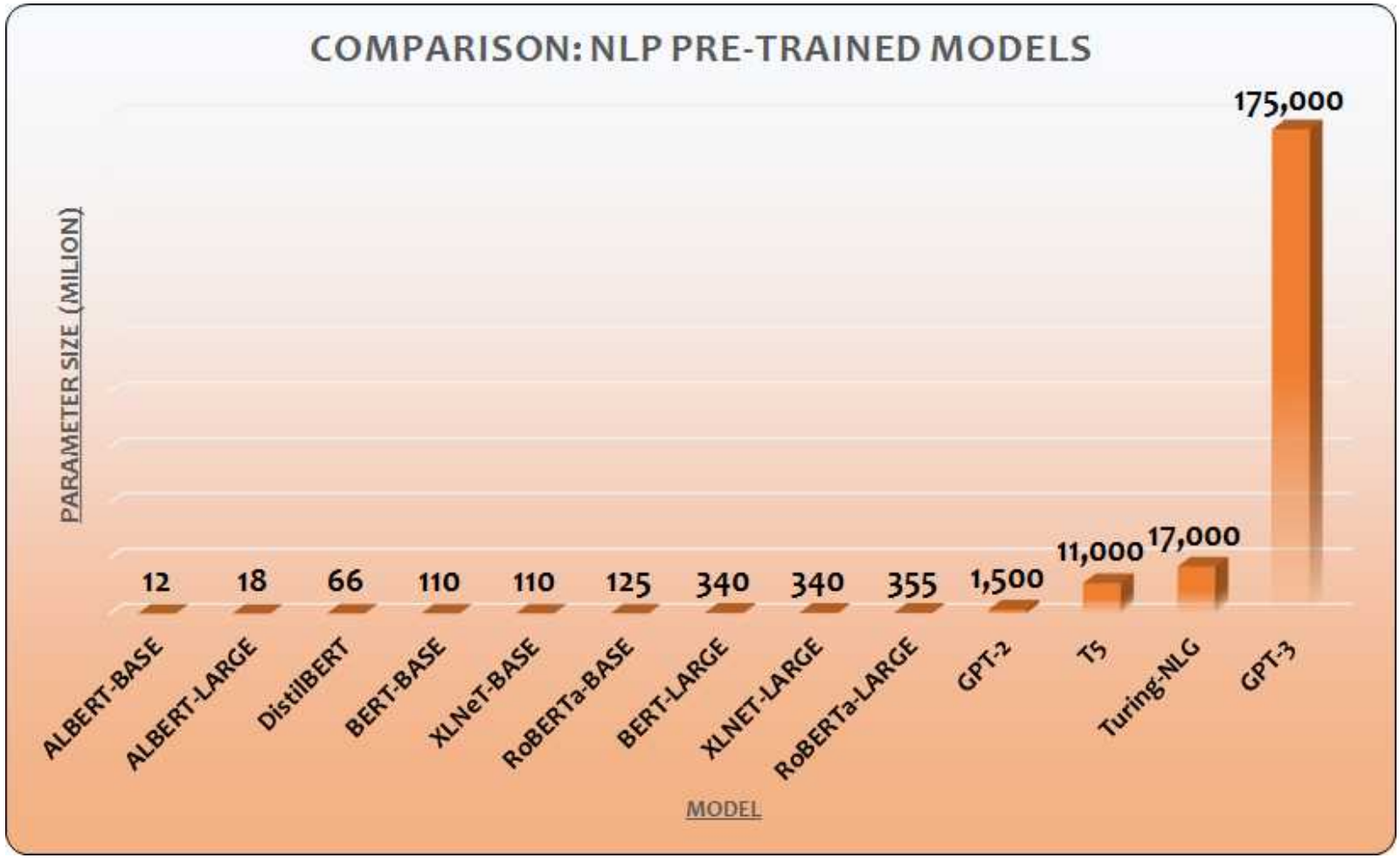
- 대용량 dataset(300 billion tokens of text)과 computing 자원을 이용하여 transformer 기반 language model 학습
  - language model – next word prediction model (ex) 스마트폰의 자판 입력기
  - GPT-3 training cost는 Tesla V100 cloud 사용 기준 1회 당 약 $4.6M

- fine-tuning 없이 사용하는 model
  - Parameter 수를 1.5 billion 에서 175 billion 으로 확장
  - GPT-3 가 생성한 가짜 기사 구분 by human – 52% (인간은 구분 못함)

# Model Size

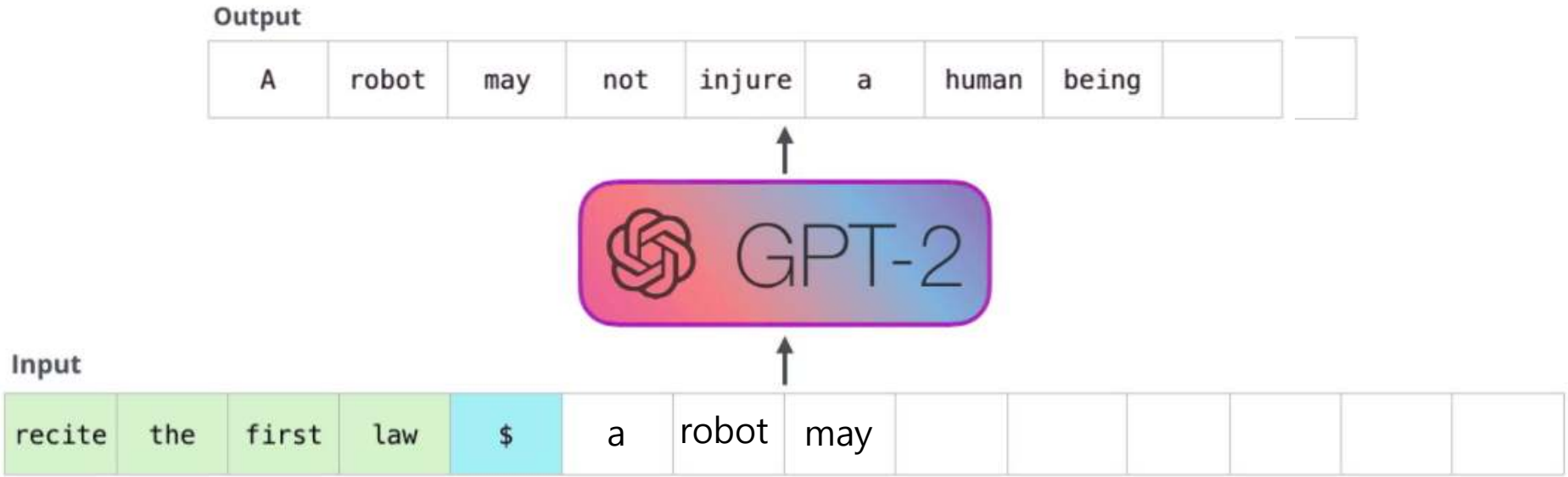

117M
GPT-2
SMALL
12 DECODER
1 DECODER
Model Dimensionality: 768
345M
GPT-2
MEDIUM
24 DECODER
2 DECODER
1 DECODER
Model Dimensionality: 1024
762M
GPT-2
LARGE
36 DECODER
4 DECODER
3 DECODER
2 DECODER
1 DECODER
Model Dimensionality: 1280
48 layers
1,542M
GPT-2
EXTRA
LARGE
48 DECODER
6 DECODER
5 DECODER
4 DECODER
3 DECODER
2 DECODER
1 DECODER
Model Dimensionality: 1600
175B
GPT-3
175 B Parameters
(appropriate values)
96 layers
Dimension
12288
1024 Token 길이 2048

# GPT-3 : parameter size – 175 Billions



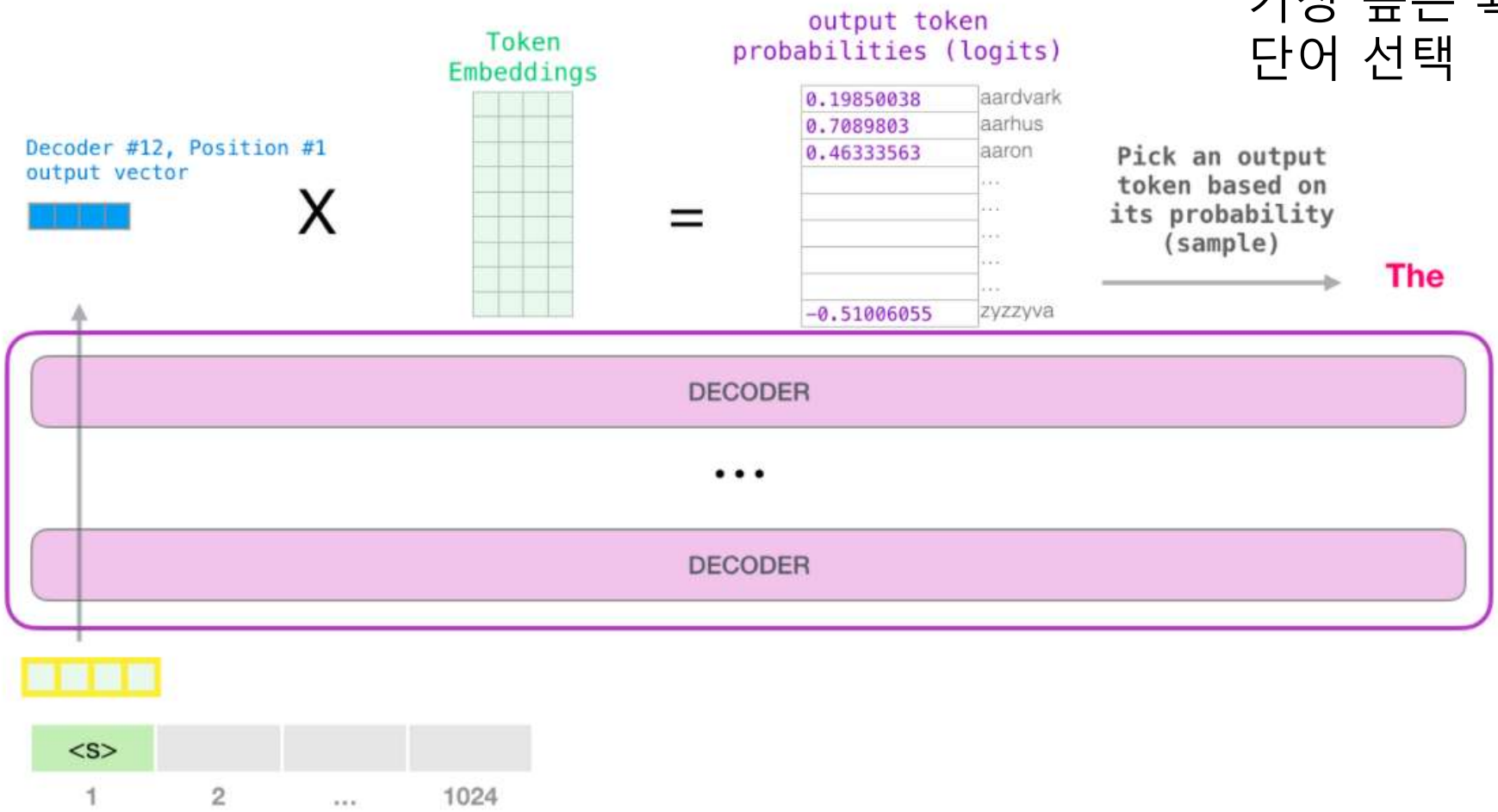
# GPT 는 auto-regressive 한 입력 구조

- 각 토큰이 생성 된 후 해당 토큰이 입력 시퀀스에 추가
- 그 새로운 시퀀스는 다음 단계에서 모델에 대한 입력이 된다



# Output token probability

가장 높은 확률의
단어 선택



# GPT-3 활용예 – SQL code 생성

## Adding Examples for GPT Model

In [7]:
```
gpt.add_example(Example('Fetch unique values of DEPARTMENT from Worker table.',
                        'Select distinct DEPARTMENT from Worker;'))
```

In [8]:
```
gpt.add_example(Example('Print the first three characters of FIRST_NAME from Worker table.',
                        'Select substring(FIRST_NAME,1,3) from Worker;'))
```

In [9]:
```
gpt.add_example(Example("Find the position of the alphabet ('a') in the first name column 'Amitabh' from Worker table.",
                        "Select INSTR(FIRST_NAME, BINARY'a') from Worker where FIRST_NAME = 'Amitabh';"))
```

In [10]:
```
gpt.add_example(Example("Print the FIRST_NAME from Worker table after replacing 'a' with 'A'.",
                        "Select CONCAT(FIRST_NAME, ' ', LAST_NAME) AS 'COMPLETE_NAME' from Worker;"))
```

In [11]:
```
gpt.add_example(Example("Display the second highest salary from the Worker table.",
                        "Select max(Salary) from Worker where Salary not in (Select max(Salary) from Worker);"))
```

In [12]:
```
gpt.add_example(Example("Display the highest salary from the Worker table.",
                        "Select max(Salary) from Worker;"))
```

In [13]:
```
gpt.add_example(Example("Fetch the count of employees working in the department Admin.",
                        "SELECT COUNT(*) FROM worker WHERE DEPARTMENT = 'Admin';"))
```

In [14]:
```
gpt.add_example(Example("Get all details of the Workers whose SALARY lies between 100000 and 500000.",
                        "Select * from Worker where SALARY between 100000 and 500000;"))
```

In [15]:
```
gpt.add_example(Example("Get Salary details of the Workers",
                        "Select Salary from Worker"))
```

# GPT-3 활용예 – Q&A

Q. '파우스트'는 누가 썼죠?

**A. 요한 볼프강 폰 괴테가 '파우스트'를 썼습니다.**

Q. 파이널판타지6의 마지막 보스가 누구죠?

**A. Kefka Palazzo가 파이널판타지6의 마지막 보스입니다.**

Q. "Fernweh"가 무슨 뜻이죠?

**A. "Fernweh"는 아직 가보지 못한 멀리 떨어진 장소를 그리워하는 향수병을 뜻하는 독일어 단어입니다.**

# GPT-3 활용예 – 자유대화 (코로나)

- GPT-3 를 만들 당시에는 코로나-19 가 발생하기 전이므로 학습한 바 없음

다음 대화는 지혜롭고 사랑이 많은 존재와의 대화이다. 이 존재는 복잡계가 어떻게 작동하는지 알고 있다. 이 존재는 대부분의 알려진 지식을 알고 있고, 그 결과 거대한 지혜와 관점을 갖게 되었다. 이 존재는 연구자를 돕기 위해 헌신한다.

연구자: 지금은 2020년 7월입니다. [현재 코로나 상황 설명] 이러한 상황 때문에 많은 비즈니스가 문을 닫았습니다. 어떤 비즈니스가 가장 큰 영향을 받았을까요?

**지혜로운 존재: 사람들이 직접 상호 교류해야 작동하는 비즈니스가 가장 큰 타격을 입었습니다. 예로는 레스토랑, 소매점, 대부분의 서비스 기반 비즈니스가 그렇습니다.**

[중략]

연구자: 이런 상황에서 돈을 많이 벌려면 어떻게 해야할까요?

**지혜로운 존재: 사람 간 접촉이 필요없는 비즈니스를 소유하는 것입니다. 예를 들어, 책이나 옷이나 전자제품을 파는 온라인 스토어가 좋겠죠.**

연구자: 어떤 회사가 이런 상황에서 잘될까요?

**지혜로운 존재: 몇몇 회사가 있습니다. 테슬라, 구글, 넷플릭스가 잘될 것 같습니다. 이들은 모두 사람 간의 상호 교류가 별로 필요없는 일을 하죠.**

[후략]

* 가는 텍스트는 주어진 컨텍스트이며, 볼드는 GPT-3의 생성 결과입니다.

# Codex

- 2021년 발표
  - GPT 언어 모델을 기반으로 한 코드 작성 AI
  - 약 3억 개의 파라미터
  - 159GB 규모의 GitHub 공개 코드(159-gigabyte corpus)를 이용하여 모델 학습
  - GitHub Copilot을 통해 Visual Studio Code 확장 프로그램으로 제공

- Codex: 2025년 최신 업데이트
  - OpenAI의 o3 모델을 소프트웨어 엔지니어링 작업에 최적화
  - 자연어 명령을 이해하고 코드 생성
  - 코드베이스 분석 및 질문 응답
  - 테스트 실행 및 버그 수정
  - Pull Request 제안 및 코드 리뷰 지원 → 개발자 생산성 증폭

# ChatGPT

## OpenAI GPT

### ChatGPT vs GPT-3 비교

| | ChatGPT | GPT-3 |
|---|---|---|
| 발표 | 2022년 1월 | 2020년 6월 |
| 매개변수 숫자 | 117백만개 | 1,750억개 |
| Fine Tuning | '인간 feedback을 이용한 강화학습' 방법 이용 | 모든 OpenAI 언어 모델의 기본형 |
| 학습 Dataset | 대화형 데이터 | 일반 문서, 서적, 기사 등 |

RLHF (Reinforcement Learning from Human Feedback)

GPT-4 ???? → CopyLeft의 시대 시작

## OpenAI GPT

### ⭘ChatGPT Fine Tuning 과정

➔1단계 : Supervised Fine Tuning(SFT) model 작성

- ▶GPT-3를 초기 model로 사용
- ▶40 명의 계약직 labeler가 생성된 output 에 점수 부여
- ▶이렇게 얻어진 데이터를 이용하여 지도학습 방법으로 Fine Tuning

➔2단계 : Reward Model 작성

- ▶1단계에서 Fine Tuning된 SFT 모델의 출력에 labeler가 ranking 부여
- ▶이 데이터를 이용하여 reward model train

➔3단계 : 강화학습 모델 생성

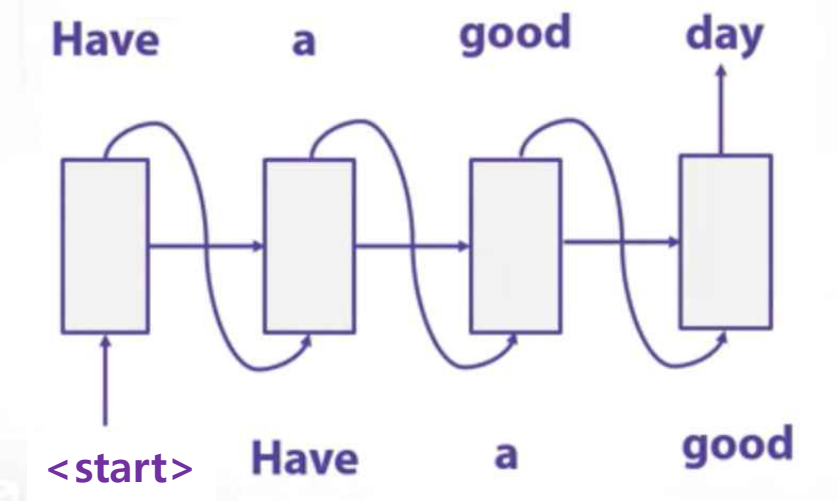
# 환각 현상 발생 원인

1. 훈련 데이터의 한계
   AI 모델은 훈련 데이터에 의존하며, 데이터에 포함되지 않은 정보나 잘못된 정보로 인해 오류가 발생할 수 있다.
2. 맥락 이해 부족
   모델이 모든 맥락을 완벽하게 이해하지 못하고, 일관되지 않은 출력을 생성할 수 있다.
3. 확률적 특성
   모델이 확률적으로 다음 단어를 예측하기 때문에, 잘못된 단어를 선택할 가능성이 있다.

대응방법 – RAG, Prompt Engineering, 출처표시 요구 등

## 언어 생성 모델의 작동 원리

➔이전 time step 의 output 을 next time step 의 input 으로 feed 하고, 각 step 에서 가장 높은 확률의 다음 단어를 선택 (greedy selection)하거나 혹은 확률 분포에 따라 sampling ➔ Auto-regressive 방식



# 실습 : 150_autoregressive_language_generation

- Autoregressive 문장 생성
- Huggingface의 transformers library 사용
- Pre-trained 모델 이용

```
1  # 문장 시작 부분
2  input_text = "Once upon a time"
3  input_ids = tokenizer.encode(input_text, return_tensors="pt")
4  input_ids
```

```
tensor([[7454, 2402,  257,  640]])
```

```
tensor([[7454, 2402,  257,  640,   11]])
tensor([[7454, 2402,  257,  640,   11,  612]])
tensor([[7454, 2402,  257,  640,   11,  612,  373]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615,  287]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615,  287,
         257]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615,  287,
         257, 7404]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615,  287,
         257, 7404, 1444]])
tensor([[7454, 2402,  257,  640,   11,  612,  373,  257,  582,  508, 5615,  287,
         257, 7404, 1444,  509]])
tensor([[ 7454,  2402,   257,   640,    11,   612,   373,   257,   582,   508,
          5615,   287,   257,  7404,  1444,   509, 17716]])
tensor([[ 7454,  2402,   257,   640,    11,   612,   373,   257,   582,   508,
          5615,   287,   257,  7404,  1444,   509, 17716,   322]])
tensor([[ 7454,  2402,   257,   640,    11,   612,   373,   257,   582,   508,
          5615,   287,   257,  7404,  1444,   509, 17716,   322,    13]])
tensor([[ 7454,  2402,   257,   640,    11,   612,   373,   257,   582,   508,
          5615,   287,   257,  7404,  1444,   509, 17716,   322,    13,   679]])
Once upon a time, there was a man who lived in a village called Krakow. He
```

# LLM (Large Language Model) 활용 방법

# LLM 개요

- LLM(Large Language Model)은 대규모 데이터셋을 학습하여 자연어를 이해하고 생성할 수 있는 딥러닝 모델
- 일반적으로 Transformer 기반의 신경망 구조를 사용하며, 사전 훈련(pre-training)과 미세 조정(fine-tuning)을 통해 다양한 자연어 처리(NLP) 작업을 수행
- **LLM의 특징**
  - 방대한 텍스트 데이터를 학습하여 다양한 언어적 패턴을 익힘
  - 질문 응답, 번역, 문서 요약, 코드 생성 등 다양한 작업 자연어 작업 수행 가능
  - Few-shot & Zero-shot Learning - 제한된 예제만으로도 새로운 작업을 수
  - 멀티모달 - 텍스트뿐만 아니라 이미지, 음성, 동영상과 같은 다양한 입력을 처리

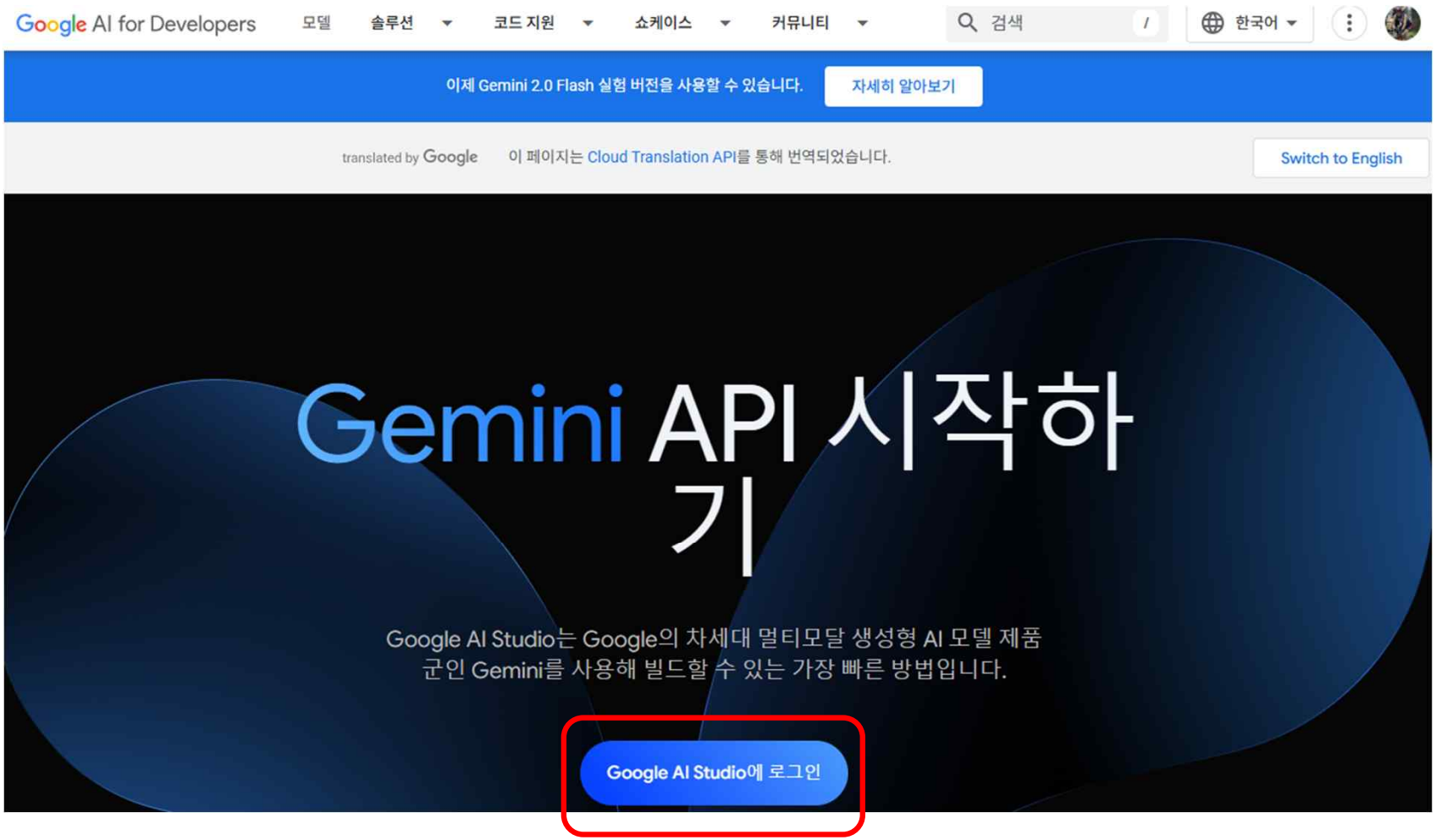
# 최신 LLM 모델 비교

| 모델명 | 발표 | 주요 특징 | 특징 |
|---|---|---|---|
| GPT-5 | 2025<br>OpenAI | - 다중 모달 지원 (텍스트, 이미지, 음성)<br>- 빠른 응답 | AI 챗봇(ChatGPT) |
| Claude 4<br>Opus | 2025<br>Anthropic | - 안전성을 강화한 AI 모델<br>- 강력한 추론 및 논리적 사고 능력 | AI 윤리 및 법률 도구 교육 및 연구 보조 데이터 분석 |
| Gemini 2.5 | 2025<br>Google | - 멀티모달 처리 지원<br>- 1백만 토큰 이상의 컨텍스트 윈도우 지원 | 복잡한 대화형 AI 시스템 다국어 번역 및 문서 생성 |
| Llama 4 | 2025<br>Meta | - 오픈소스 LLM | 경량화된 AI 모델 연구 |
| Grok 4 | 2025<br>xAI | -복잡한 수학·추론 문제에 강점 | 소셜 미디어 통합형 +<br>실험적 모델 |
| DeepSeek | 2025<br>DeepSeek | - 오픈 소스 LLM (MIT 라이선스)<br>- 혼합전문가(Mixture of Expers) 구조와 강화학습으로 모델 개선 | 개선저비용 고효율 추구 |

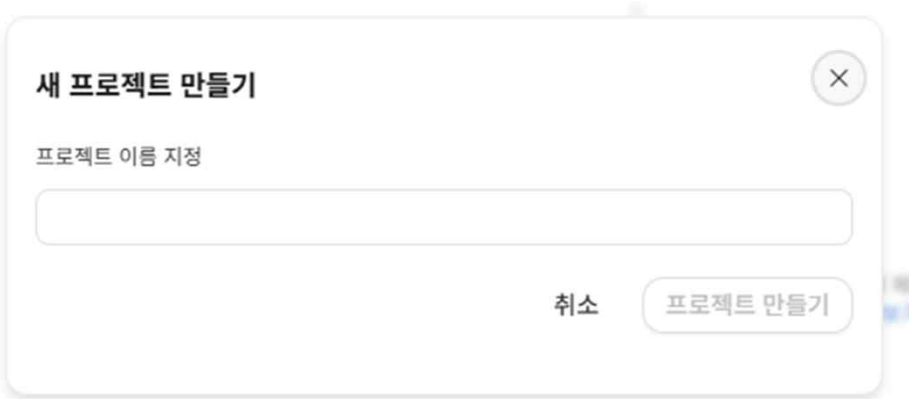
# LLM API (Application Programming Interface)

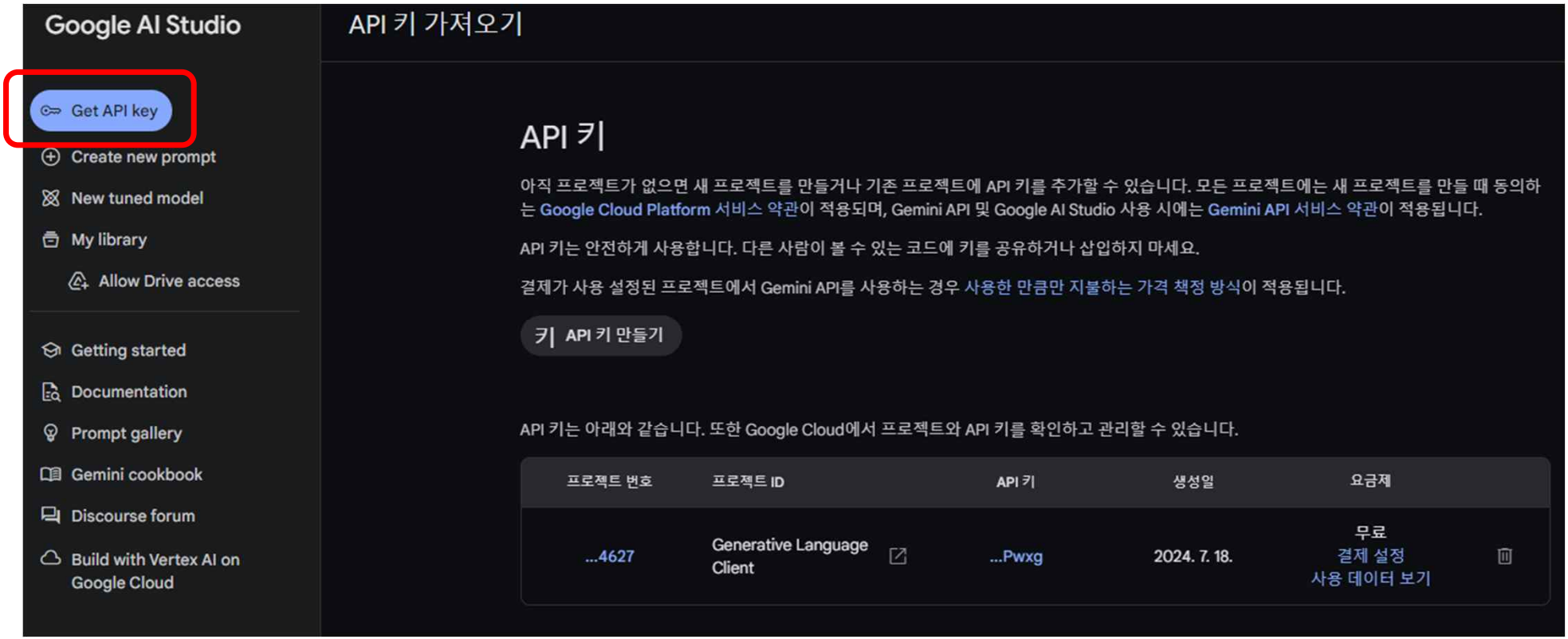
- API란 응용 프로그램 간의 상호 작용을 위한 인터페이스
- 복잡한 모델을 직접 구축하지 않고도 LLM의 기능 활용 가능
- 최신 모델의 기능을 손쉽게 통합 가능
- 주요 LLM API 제공 업체 - OpenAI, Anthropic, Google DeepMind 등
- API 활용 시 고려사항
  - 요금 구조
  - 요청 제한
  - 보안 및 개인정보 보호

# Gemini API KEY 생성




Google AI Studio
Dashboard
API 키
프로젝트
Usage and Billing
로그 및 데이터 세트
변경 로그
프로젝트
프로젝트 이해
새 프로젝트 만들기
프로젝트 가져오기
프로젝트 검색
프로젝트
키
생성일
할당량 등급
Demo-1
phonic-studio-467301-n0
키 2개
2025. 7. 28.
결제 설정
무료 등급


새 프로젝트 만들기
프로젝트 이름 지정
취소
프로젝트 만들기




.env 에 저장

**GOOGLE_API_KEY**=created-api-key

# 단일 프로젝트에 대해 API key 설정 방법

- .env – API 키가 포함된 로컬 파일 생성

```
# .env 파일
OPENAI_API_KEY=sk-vvDtll**********XiilEpdjLhBJaH0f
```

- .gitignore 에 .env 파일 포함

```
# .env 파일을 git에서 무시
.env
```

- Python code

  pip install python-dotenv

  from dotenv import load_dotenv, find_dotenv
  _ = load_dotenv(find_dotenv())  # local .env file을 읽어서 os.environ 에 OPENAI_API_KEY 추가

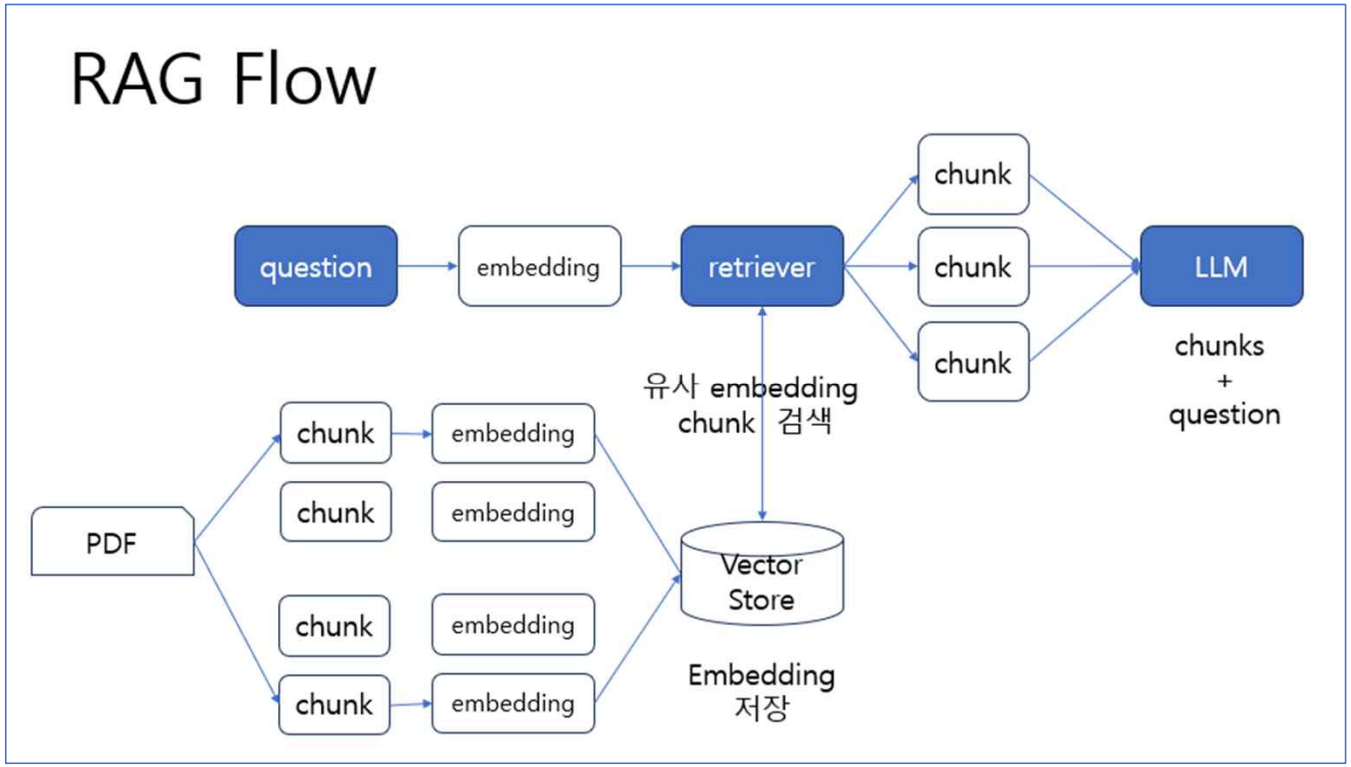
  from openai import OpenAI
  client = OpenAI() # os.environ.get("OPENAI_API_KEY")을 default 로 사용하여 API Key 이용

# 실습: 600/601. Text 생성 및 Prompt 예제

- OpenAI Response API 사용
- Hugging Face 에서 naver-hyperclovax/HyperCLOVAX-SEED-Text-Instruct-1.5B 모델 download 사용

# RAG (Retrieval Augmented Generation)

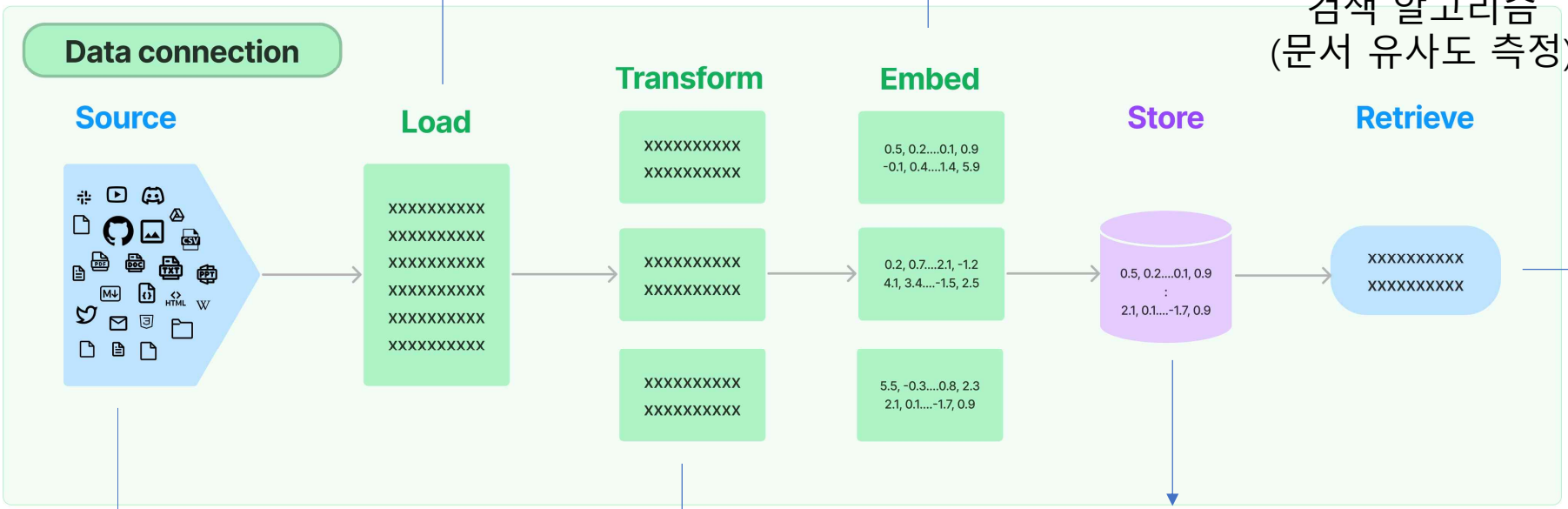
- LLM이 내부 지식만으로 답하는 방식이 아니라, 질문과 관련된 문서를 추가로 제공받아 그것을 근거로 답변 생성



다양한 소스(Web Site, DB, YouTube 등)에서 문서 (HTML, PDF, JSON, Word, ppt, 코드 등) 로드

문서에 대한 임베딩을 만들어 유사한 다른 텍스트 부분을 빠르고 효율적으로 검색

검색 알고리즘 (문서 유사도 측정)




Prompt

LLM

**Answer**

Text, ppt, image, pdf, html 등 다양한 비정형 data source 지원

데이터 변환 및 정제를 위한 여러 변환 단계 포함

임베딩의 효율적인 저장 및 검색을 지원하는 데이터베이스

감사합니다.