

자연어 처리

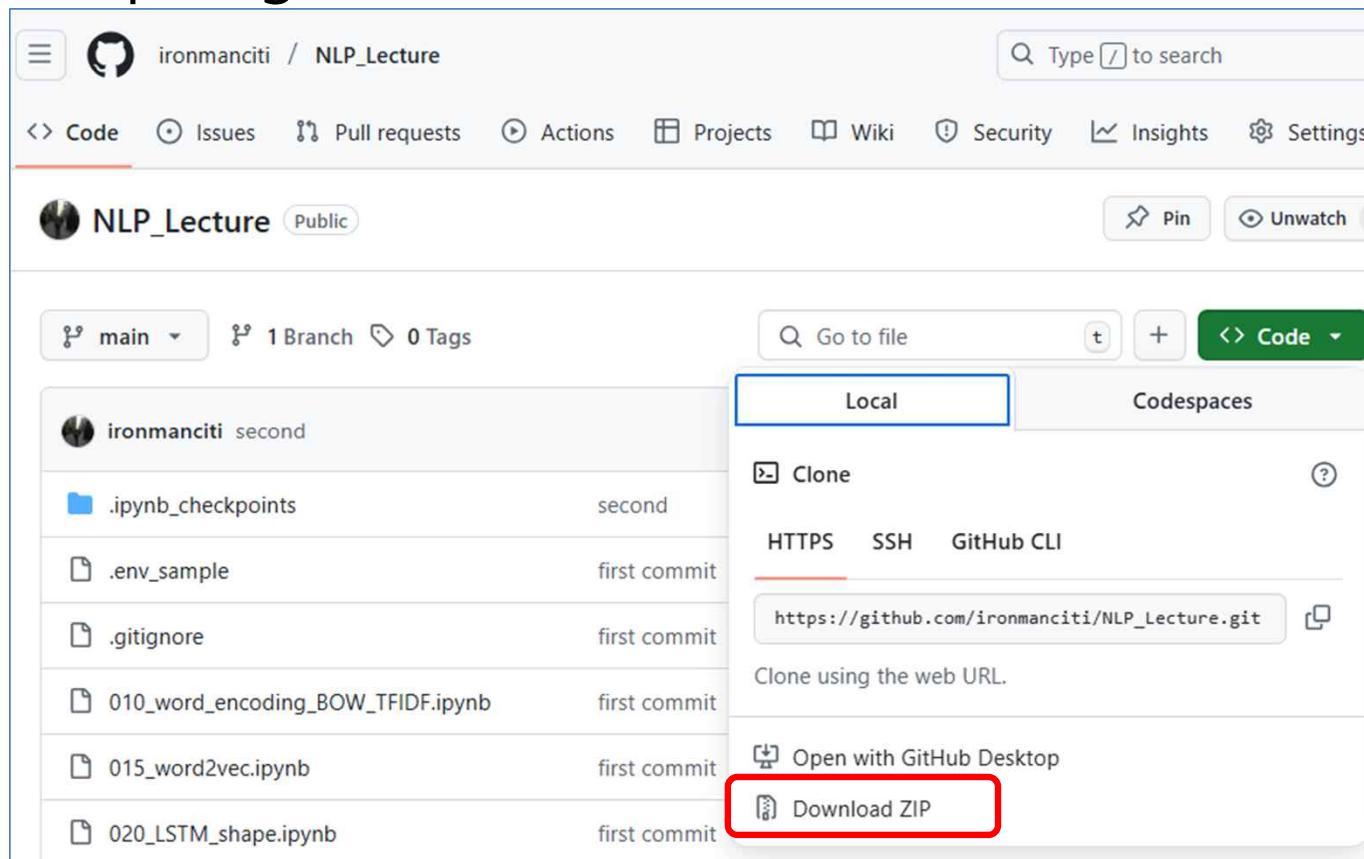
강사: 오영제

교육 환경 준비

- 인터넷 연결
- Chrome Browser
- Gmail ID for Colab

Github Repository 에서 실습 code download

https://github.com/ironmanciti/NLP_Lecture

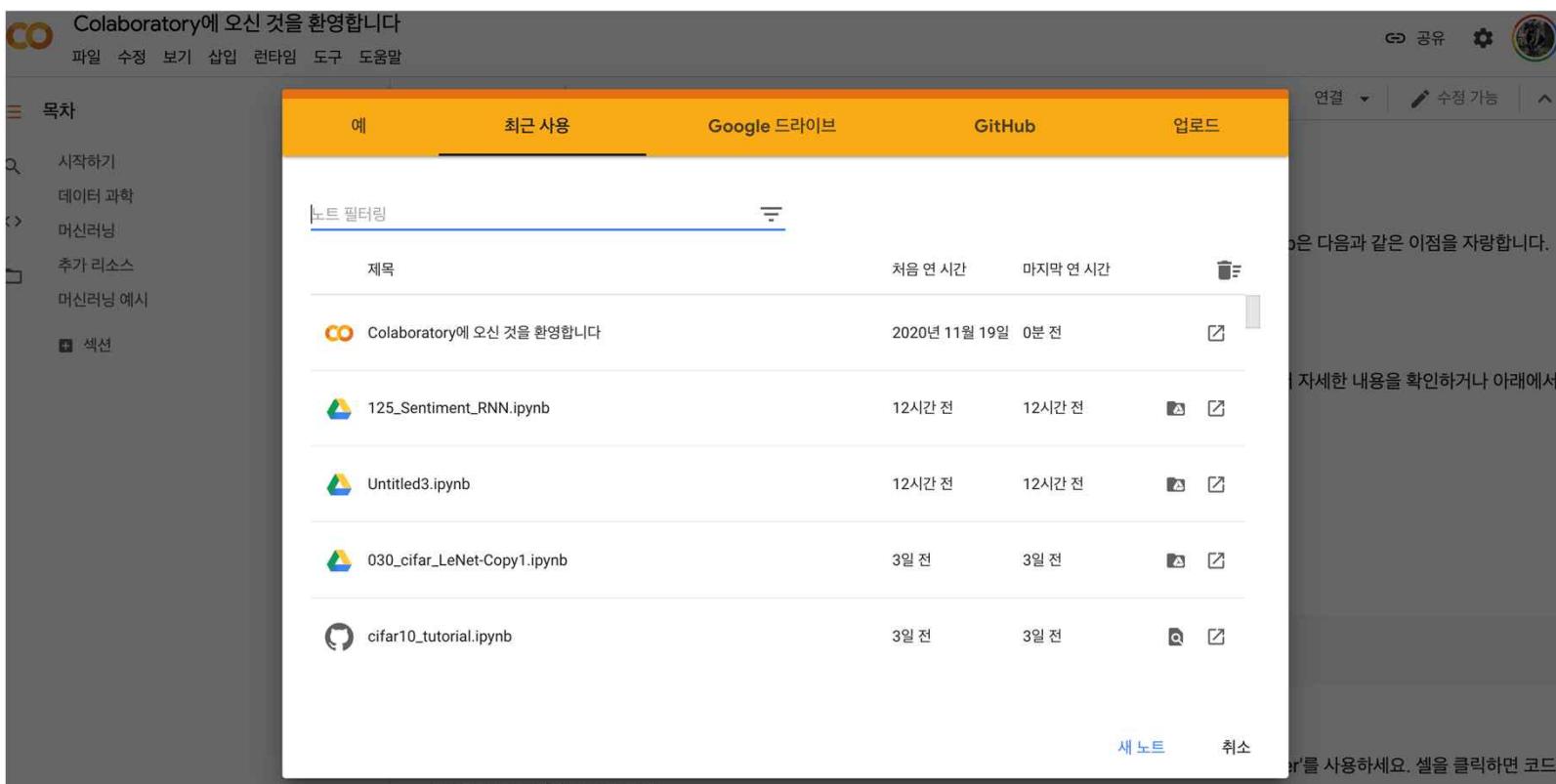


Google Colaboratory 소개

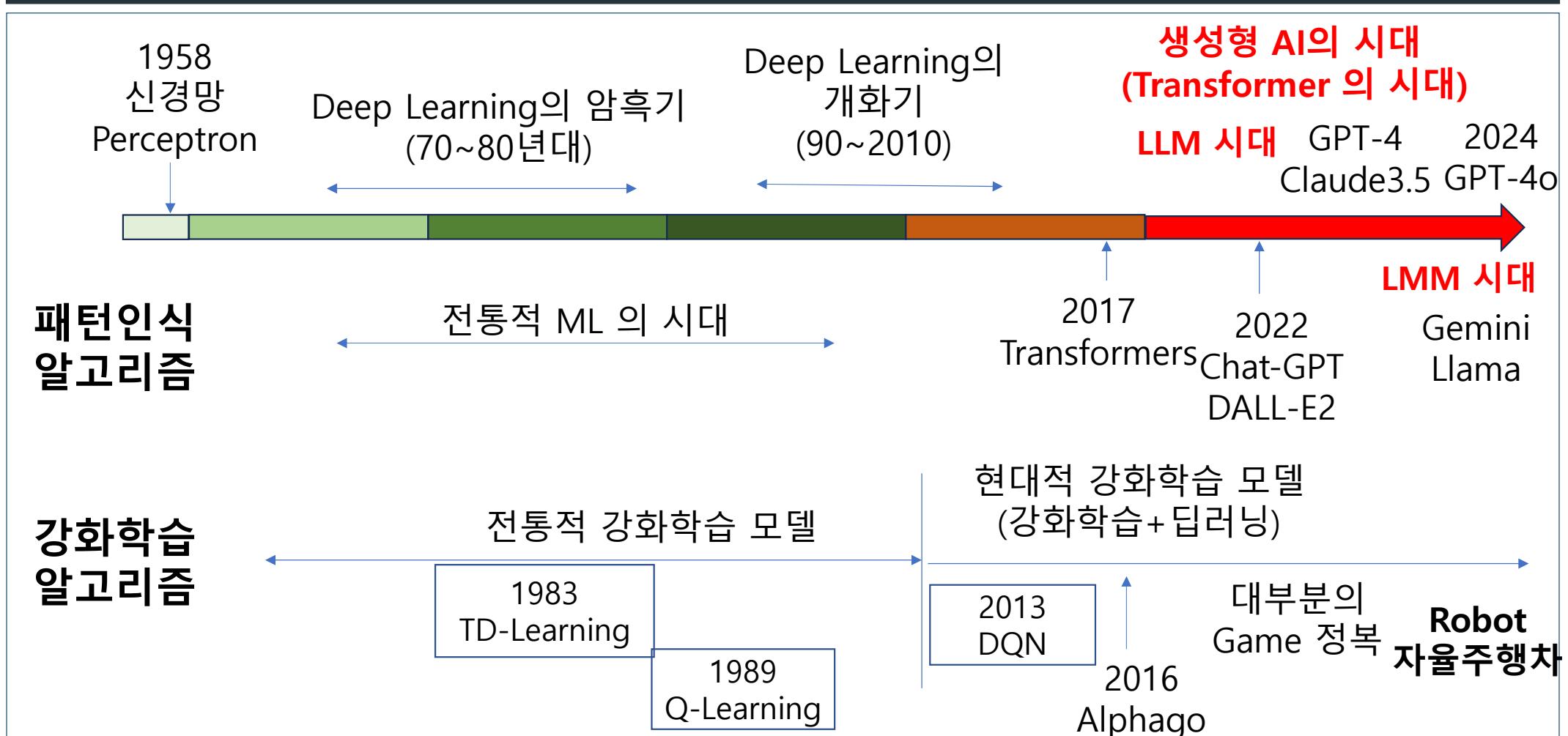
- Free GPU 제공
- Google Drive 와 연동
- Jupyter Notebook 환경
- Deep Learning beginner 를 위한 최적의 환경
- 각종 snippet 제공

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



Advertisement Matching



Spell Checking



Keyword Search

Machine Translation



Sentimental Analysis



Chatbot

Speech Recognition



Text Generation



Q&A

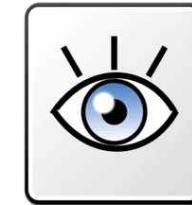


Image Captioning



Video Captioning



Text Summarization



Topic Modeling

NLP 기술 적용 분야

- 자연어 이해 (Natural Language Understanding)

- Sentiment Analysis (감성분석) - ex) 이 영화평이 긍정적인가 부정적인가 ?
- Topic modeling (주제분류) - ex) news 내용을 정치, 스포츠, 과학등으로 분류
- 형태소 분석
- 개체명 인식 (Named Entity Recognition)
- 철자법 교정 (Spelling correction)
- Spam Detection : email 이 spam 인지 ham 인지 구분

- 자연어 생성 (Natural Language Generation)

- Text generation (문장 생성)
- 번역
- 대화

발전 단계

Resolved

스팸분류
품사결정
개체명 인식
기본 문서 생성
간단한 질의 응답

Mostly Resolved

감성분석
구문분석
기계번역
정보추출
요약
이미지-텍스트 통합

Making Good Progress

복잡한 질의응답
의역
대화
비디오-텍스트 통합
다중모달 감성 분석
창의적 문서 생성

전통적 기법 (규칙기반, 통계기반)

딥러닝 기법



대부분의 자연어 처리 문제는 “분류” 문제

NLP 에 대한 접근 방법 (Language Modeling)

기존 방식 (Data-Driven Approach)	최근 방식 (Neural Network)
<ul style="list-style-type: none">• 2010 년대 이전 전통적 방식• 언어 전문가에 의한 고품질 sample data 확보가 중요• 규칙기반 (Rule-Based)<ul style="list-style-type: none">• 언어학을 기반으로 한 rule-based program• 통계기반 (Statistical Machine Translation)<ul style="list-style-type: none">• 말뭉치 (Corpus) 를 기반으로 통계 모델 및 전통적 machine learning 적용	<ul style="list-style-type: none">• Deep Learning 이용• Word Embedding 을 기반으로 전체 입력 문장 단위로 처리• 어순, 단어의 의미, 문맥 파악 등을 스스로 학습• 언어 전문가 불필요• Word Embedding• Bidirectional Recurrent Neural Network• Encoder-decoder (seq2seq) model• Attention model• Transformer model• BERT/GPT-3, etc

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](#)

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

The diagram illustrates the process of text vectorization. On the left, three text documents are shown in a table:

good movie
not a good movie
did not like

An arrow points from this table to a binary matrix on the right, which represents the feature counts for each token ('good', 'movie', 'not', 'a', 'did', 'like').

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

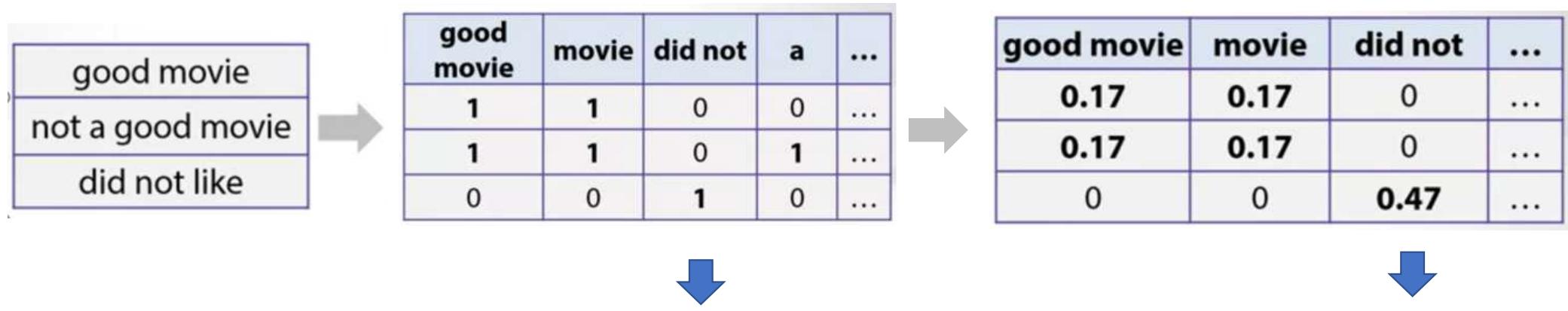
	good movie	movie	did not	a	...
Document 1	1	1	0	0	...
Document 2	1	1	0	1	...
Document 3	0	0	1	0	...

→ 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

One-Hot-Encoding 표현

Vocabulary 사전

a -1
aaron – 2
. .
apple – 456
. .
king – 4914
. .
orange – 6257
. .
queen – 7157
. .
zebra – 9,999
<UNK> - 10,000

One-Hot encoding



해당되는 index 위치만 1

King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}$

10,000

Word encoding – Token 분리, word index 작성

```
1 import tensorflow as tf  
2 from tensorflow import keras  
3 from tensorflow.keras.preprocessing.text import Tokenizer
```

```
1 sentences = ['I love my dog.', 'I love my cat.', 'You love my dog!']  
2  
3 tokenizer = Tokenizer(num_words=100) # take top 100 words from the sentences  
4 tokenizer.fit_on_texts(sentences)  
5 word_index = tokenizer.word_index
```

```
1 print(word_index)
```

{'love': 1, 'my': 2, 'i': 3, 'dog': 4, 'cat': 5, 'you': 6} → 소문자 변환, 구둣점 제거

Text to sequence – 수열 (sequence) 변환

```
1 sentences = ['I love my dog.',  
2             'I love my cat.',  
3             'You love my dog!',  
4             'Do you think my dog is amazing?']  
5  
6 tokenizer = Tokenizer(num_words=100)    # take top 100 words from the sentences  
7 tokenizer.fit_on_texts(sentences)  
8 word_index = tokenizer.word_index  
9  
10 sequences = tokenizer.texts_to_sequences(sentences)
```

Corpus

```
1 print(word_index)  
2 print(sequences)
```

```
{'my': 1, 'love': 2, 'dog': 3, 'i': 4, 'you': 5, 'cat': 6, 'do': 7, 'think': 8, 'is': 9, 'amazing': 10}  
[[4, 2, 1, 3] [4, 2, 1, 6], [5, 2, 1, 3], [7, 5, 8, 1, 3, 9, 10]]
```

word index 에 없는 단어를 가진 문장 처리

```
1 test_data = ['I really love my dog',  
2             'my dog loves my lizard']  
3  
4 test_seq = tokenizer.texts_to_sequences(test_data)  
5  
6 print(test_seq)  
7 print(word_index)
```

```
[4, 2, 1, 3], [1, 3, 1]  
{'my': 1, 'love': 2, 'dog': 3, 'i': 4, 'you': 5, 'cat': 6, 'do': 7, 'think': 8, 'is': 9, 'amazing': 10}
```

my dog my



- Word index 작성 시 Large corpus 필요
- 없는 단어의 자리를 special token (oov_token) 으로 채움

```
1 sentences = ['I love my dog.',  
2             'I love my cat.',  
3             'You love my dog!',  
4             'Do you think my dog is amazing?']  
5  
6 tokenizer = Tokenizer(num_words=100, oov_token='<OOV>') # take top 100 words from the sentences  
7 tokenizer.fit_on_texts(sentences)  
8 word_index = tokenizer.word_index  
9  
10 sequences = tokenizer.texts_to_sequences(sentences)
```

```
1 print(word_index)  
2 print(sequences)
```

```
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}  
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]
```

```
1 test_data = ['I really love my dog',  
2             'my dog loves my lizard']  
3  
4 test_seq = tokenizer.texts_to_sequences(test_data)  
5  
6 print(test_seq)  
7 print(word_index)
```

```
[[5, 1, 3, 2, 4], [2, 4, 1, 2, 1]]  
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}
```

padding

- 입력 text 를 동일한 길이로 맞추기

```
4 from tensorflow.keras.preprocessing.sequence import pad_sequences  
  
1 sentences = ['I love my dog.',  
2   'I love my cat.',  
3   'You love my dog!',  
4   'Do you think my dog is amazing?']  
5  
6 tokenizer = Tokenizer(num_words=100, oov_token='<OOV>')    # take top 100 words from the sentences  
7 tokenizer.fit_on_texts(sentences)  
8 word_index = tokenizer.word_index  
9  
10 sequences = tokenizer.texts_to_sequences(sentences)  
11 padded = pad_sequences(sequences)
```

```
1 print(word_index)  
2 print(sequences)  
3 print(padded)
```

```
{'<OOV>': 1, 'my': 2, 'love': 3, 'dog': 4, 'i': 5, 'you': 6, 'cat': 7, 'do': 8, 'think': 9, 'is': 10, 'amazing': 11}  
[[5, 3, 2, 4], [5, 3, 2, 7], [6, 3, 2, 4], [8, 6, 9, 2, 4, 10, 11]]  
[[ 0 0 0 5 3 2 4]  
[ 0 0 0 5 3 2 7]  
[ 0 0 0 6 3 2 4]  
[ 8 6 9 2 4 10 11]]
```

One-Hot-Encoding

```
1 sentence = ["I really think this is amazing. honest."]
2 sequence = tokenizer.texts_to_sequences(sentence)
3 print(sequence)
4 print()
5 print(to_categorical(sequence))
```

```
[[5, 1, 9, 1, 10, 11, 1]]
```

```
[[[0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]]
```

실습: 010-문장의 Vector 표현

- sklearn BOW/TF-IDF 비교
- keras 제공 API 를 사용한 word encoding
- sentence 의 token화 및 sequence 변환
- One-Hot-Encoding

One-hot-encoding 표현의 문제점

- 단어를 단순히 index 번호에 따라 one-hot-encoding 으로 vectorize 하므로 단어 간의 유사성을 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 vector로 표시
→ 단어나 문장을 **vector space**로 끼워 넣음 (**embedding**)
- 예를 들어, 호감(positive), 비호감(negative) 두 가지 label에 따라 관련 단어들을 두 개의 category로 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



King (4914) 의 4 dimension vector 표시

Man (5931) 의 4 dimension vector 표시

Embedding matrix (example)

학습된 300 차원 features

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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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
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

20 rows × 300 columns

Embedding Layer 를 이용한 vector 표현

- Projection Matrix

(5 x 3)

$$\begin{bmatrix} 0 & 0 & 0 & \textcolor{green}{1} & 0 \end{bmatrix} \times$$

$$\begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \textcolor{green}{10} & \textcolor{green}{12} & \textcolor{green}{19} \\ 11 & 18 & 25 \end{bmatrix}$$

$$[0 \times 17 + 0 \times 23 + 0 \times 4, 1 \times 10 + 0 \times 11, \\ 0 \times 24 + 0 \times 5 + 0 \times 6 + 1 \times 12 + 0 \times 18, \\ 0 \times 1 + 0 \times 7 + 0 \times 13 + 1 \times 19 + 0 \times 25]$$

$$\boxed{\begin{bmatrix} 10 & 12 & 19 \end{bmatrix}} \quad (1 \times 3)$$

$$\downarrow \quad x_k$$

$$W_{V \times N}$$

$$x_k^{New}$$

k 번째 단어 (One-Hot-Encoding)

- One-Hot-Vector 에 Projection Matrix(Embedding Layer) 를 곱해 새로운 vector 생성 \rightarrow 계산의 간편성

- Projection Matrix 의 k 번째 row 가 k 번째 단어에 대응하는 weight 임

Word2Vec

- 2013년 구글에서 개발한 Word Embedding 방법
- One-Hidden Layer 의 shallow network → 최초의 neural embedding model
- 매우 큰 Corpus (ex, 10억, 100 억 단어) 에서 자동 학습 (**비지도학습**)

Ex) 이사금께 충성을 맹세 하였다.

왕에게 충성을 맹세 하였다.

➔ “이사금”이 무슨 뜻인지는 모르겠지만 주변 단어 형태가 비슷하므로,
“왕”과 비슷한 의미일 것

Word2Vec 훈련 방법

- 중심단어로 주변단어를 (skip-gram), 주변단어로 중심단어를(CBOW) 예측하는 과정에서 단어를 벡터로 임베딩 하는 두가지 방법
- 임베딩 된 단어의 내적(inner product)이 코사인 유사도가 되도록 함

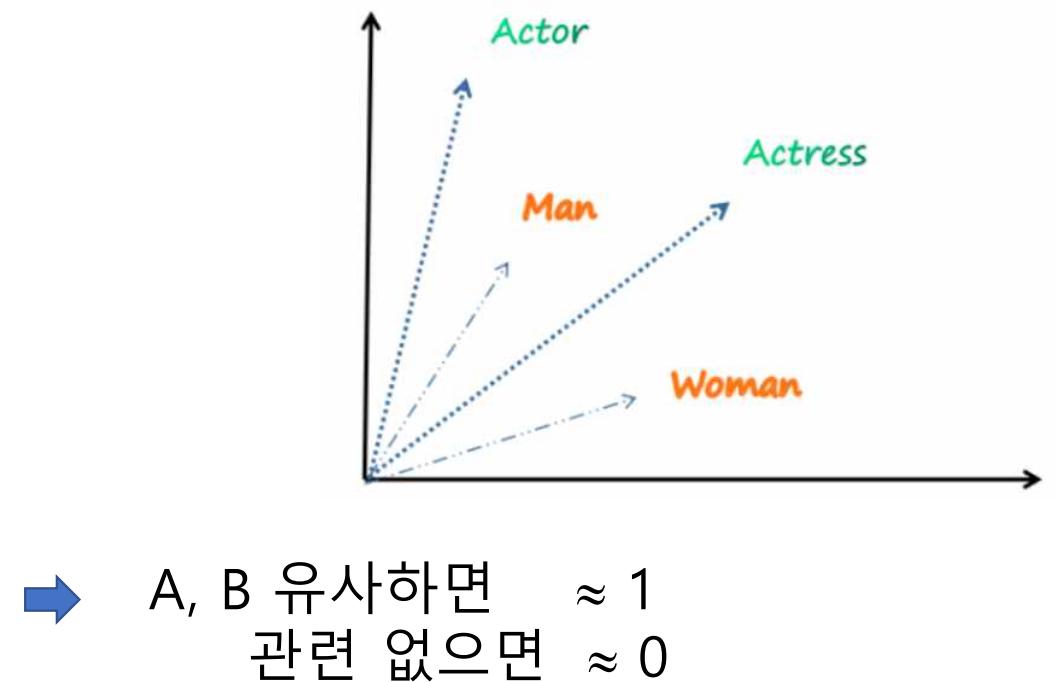
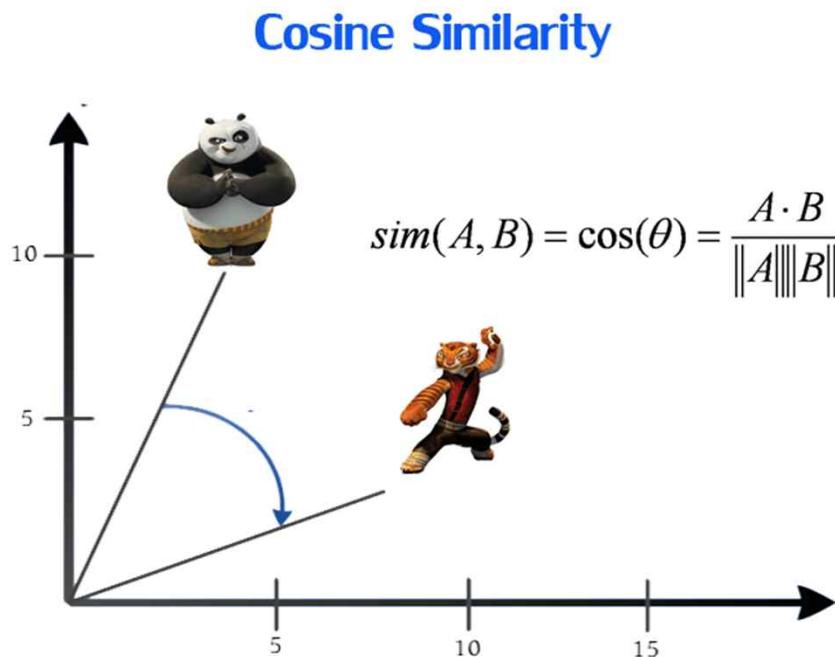
ex) I love king of Korea. (skip-gram : window size = 2)

input : king → [0,0,0,0,...1,..0]

target : I love → [0,0,0,0,...1,..0] [0,0,0,0,0,0,...1,...0]

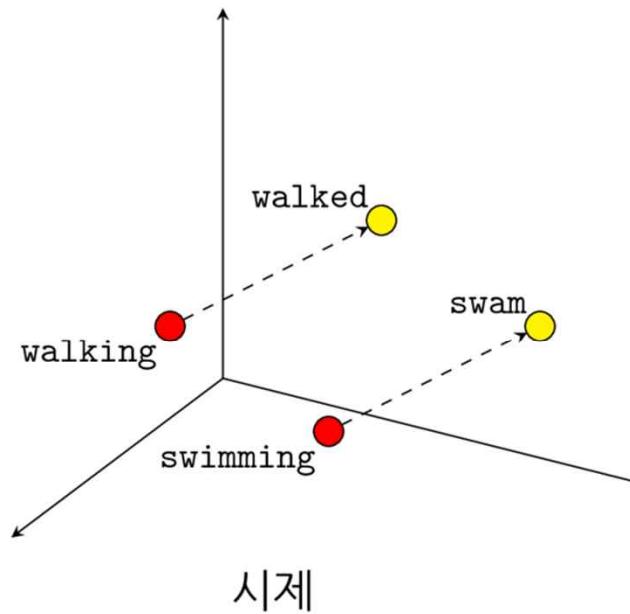
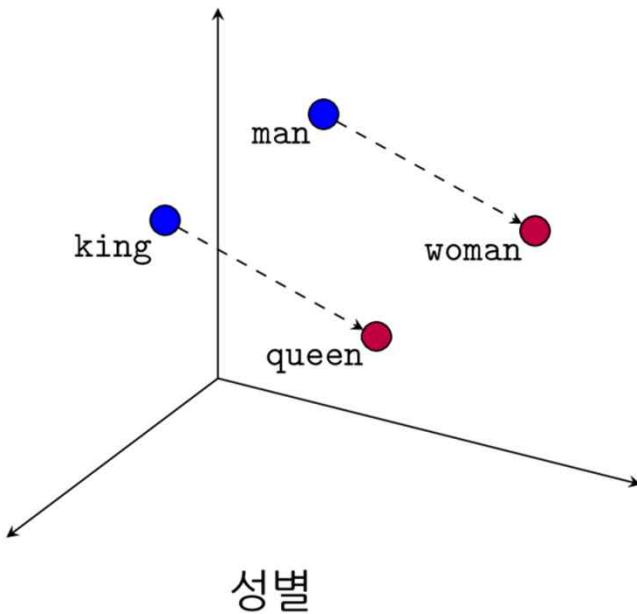
 of Korea → [0,0,0,...1,..0] [0,0,0,0,0,...1,...0]

유사도 측정 (Cosine Similarity)



Word2Vec (Word Embedding) 의 결과

king – queen ≈ man - woman



Spain	Madrid
Italy	Rome
Germany	Berlin
Turkey	Ankara
Russia	Moscow
Canada	Ottawa
Japan	Tokyo
Vietnam	Honoi
China	Beiging

국가-수도

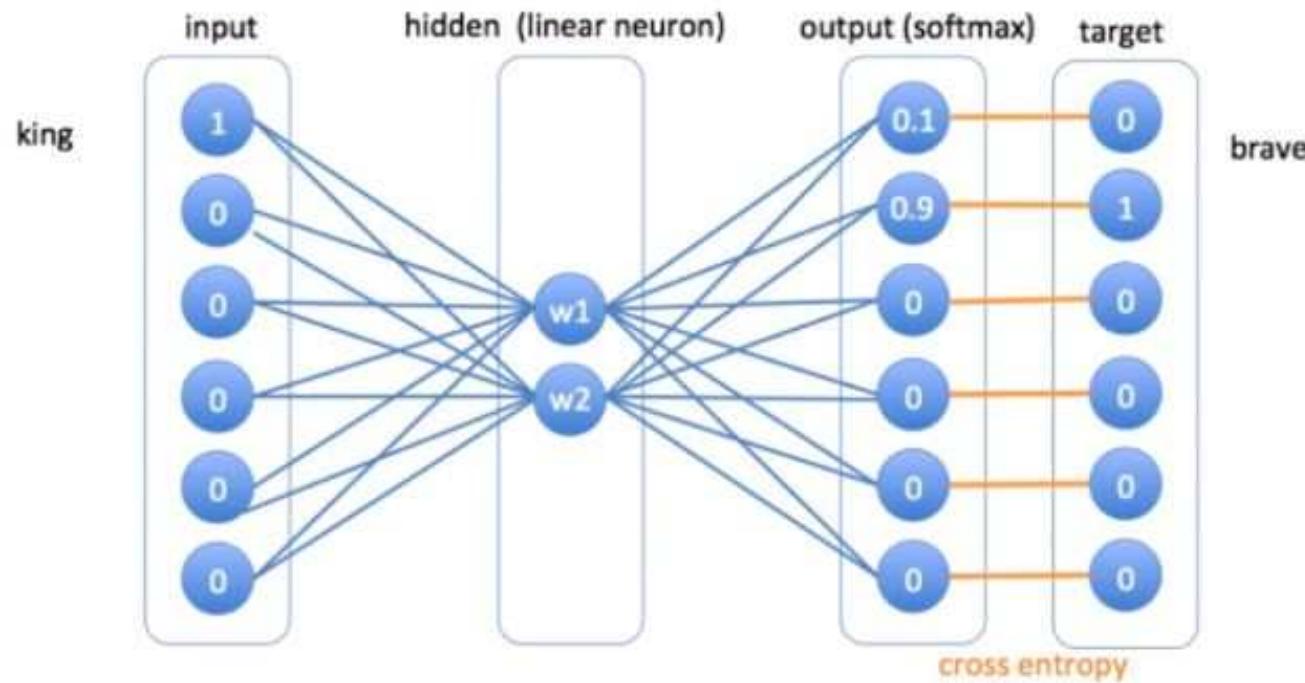
<https://ronxin.github.io/wevi/>

실습 : 015. word2vec 작성

- Skip-gram 방식 - 중심 단어(target word)를 기준으로 주변 단어(context word) 예측
- 예) window size 2 인 경우
['princess', 'young', 'woman', 'pretty', 'wise', 'queen']

중심 단어	주변 단어 (앞뒤 2개)	Skip-gram 데이터
princess	young, woman	(princess, young), (princess, woman)
young	princess, woman, pretty	(young, princess), (young, woman), (young, pretty)
woman	princess, young, pretty, wise	(woman, princess), (woman, young), (wo man, pretty), (woman, wise)

Word2vec network 구조



NLP with Deep Learning

Deep Learning

Sequence Model

What is sequence data ?

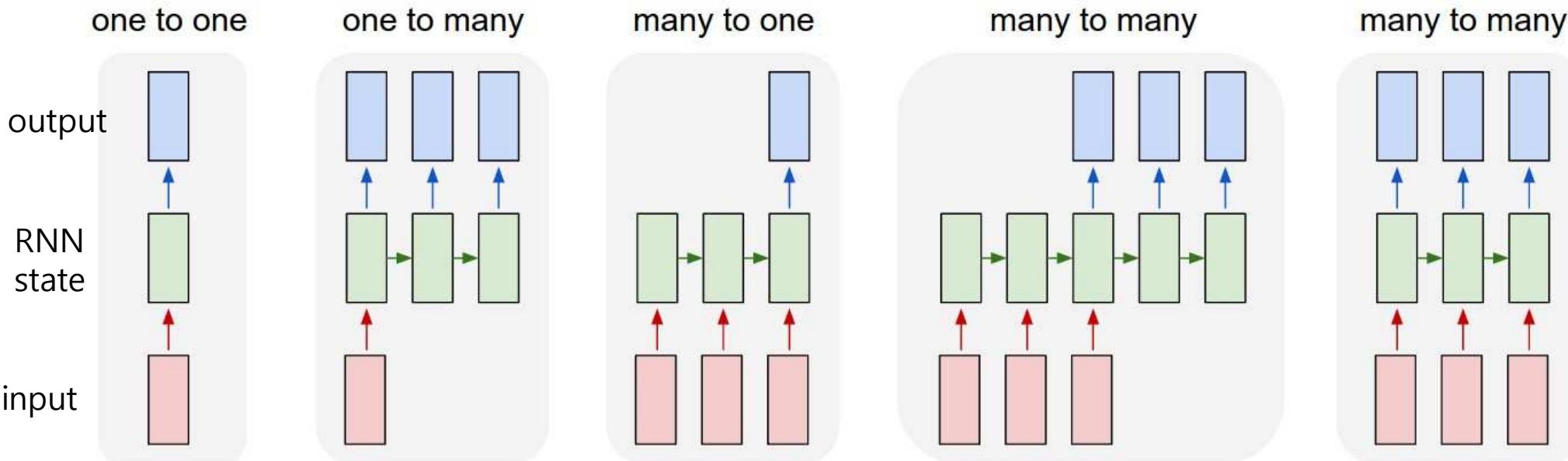
- Speech recognition : 파동의 연속 → 단어의 연속으로 변환
- Music generation : 연속된 음표 출력
- Sentiment classification : Text → 평점, 부정/긍정 판단
- DNA 분석 : 염기서열 → 질병유무, 단백질 종류 등
- 자동 번역 : 한국어 → 영어
- Video activity recognition : 연속된 장면 → 행동 판단
- Financial Data : 시계열자료 → 주가, 환율 예측 등

RNN (Recurrent Neural Network)

RNN (Recurrent Neural Network)

- 시퀀스 데이터에 특화
- '기억' 능력을 갖고 있음
 - * 네트워크의 기억 - 지금까지의 입력 데이터를 요약한 정보
(새로운 입력이 들어올 때마다 네트워크는 자신의 기억을 조금씩 수정)
- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보
(사람의 시퀀스 정보 처리 방식과 비슷, 기억을 바탕으로 새로운 단어 이해)
- 이 과정은 새로운 단어마다 계속해서 반복 → Recurrent (순환적)

Different Types of RNN



예) 이미지 분류

- 이미지로 부터 문장 생성
(Image Captioning)
- 작곡, 작시
- 감성 분석
(sentiment Analysis → positive/negative)
- Spam detection
- 기계 번역 (영어 → 한국어 문장)
- Chatbot
- Question Answering
- video 각 frame에 label 생성
- 품사 tagging
- 개체명 인식
- Character 단위 문장 생성 등

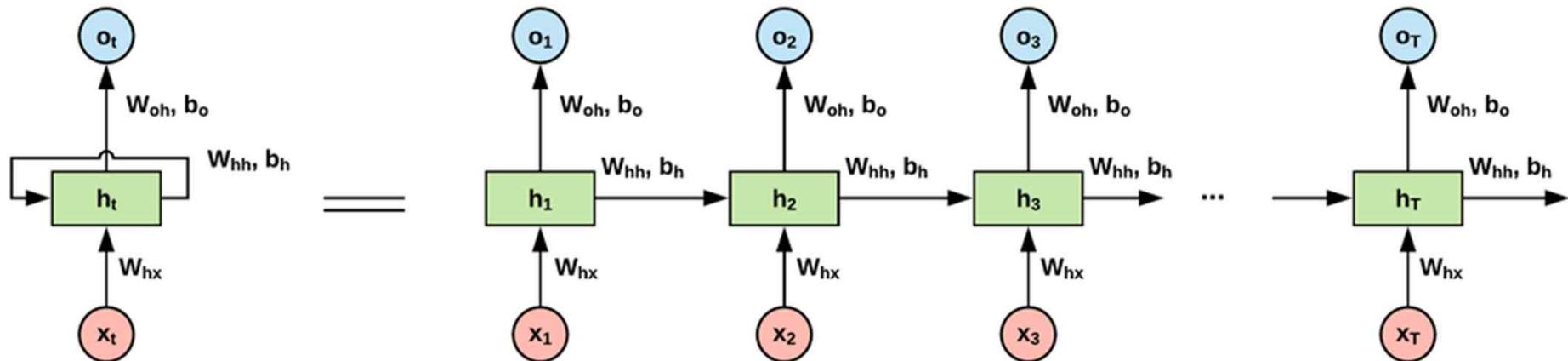
RNN (Unfold 표시)

Internal State :

$$h_t = \tanh (W_h h_{t-1} + W_x x_t)$$

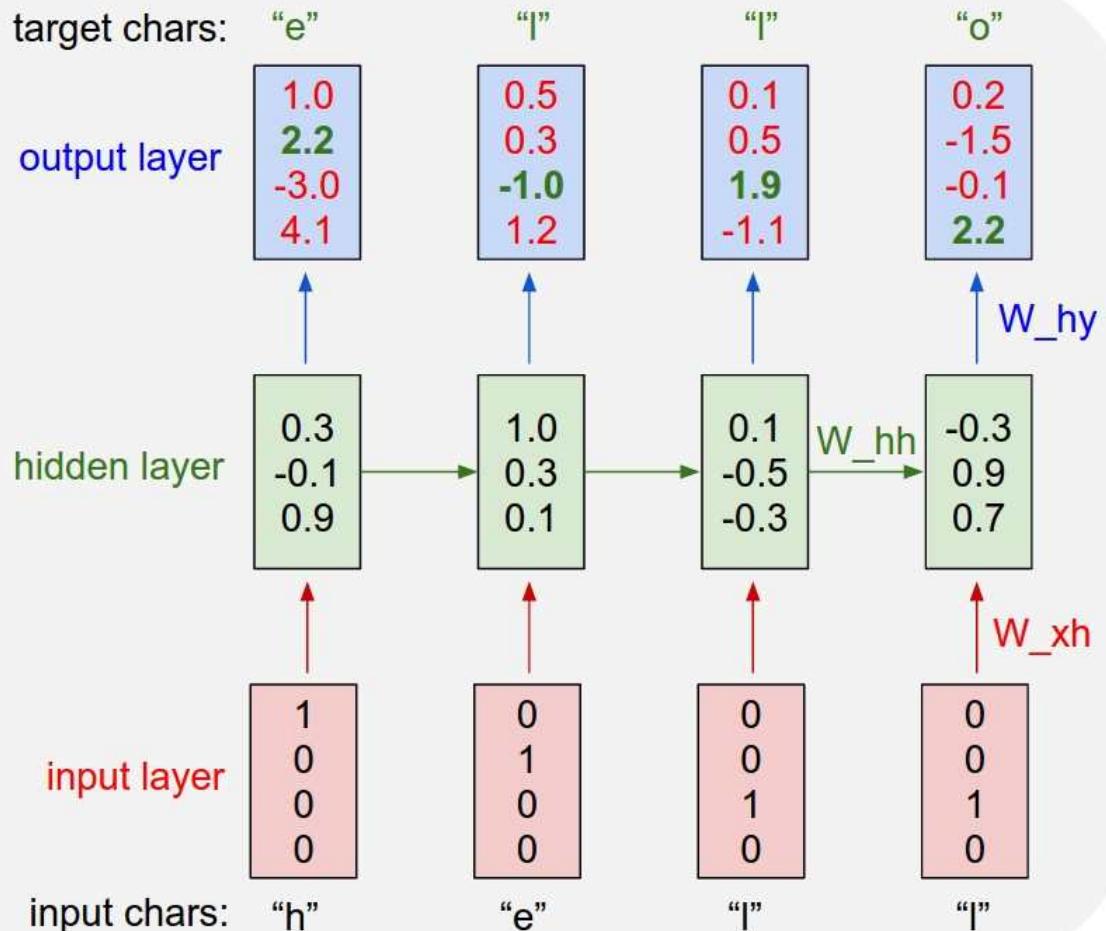
Output :

$$o_t = \text{softmax} (W_o h_t)$$



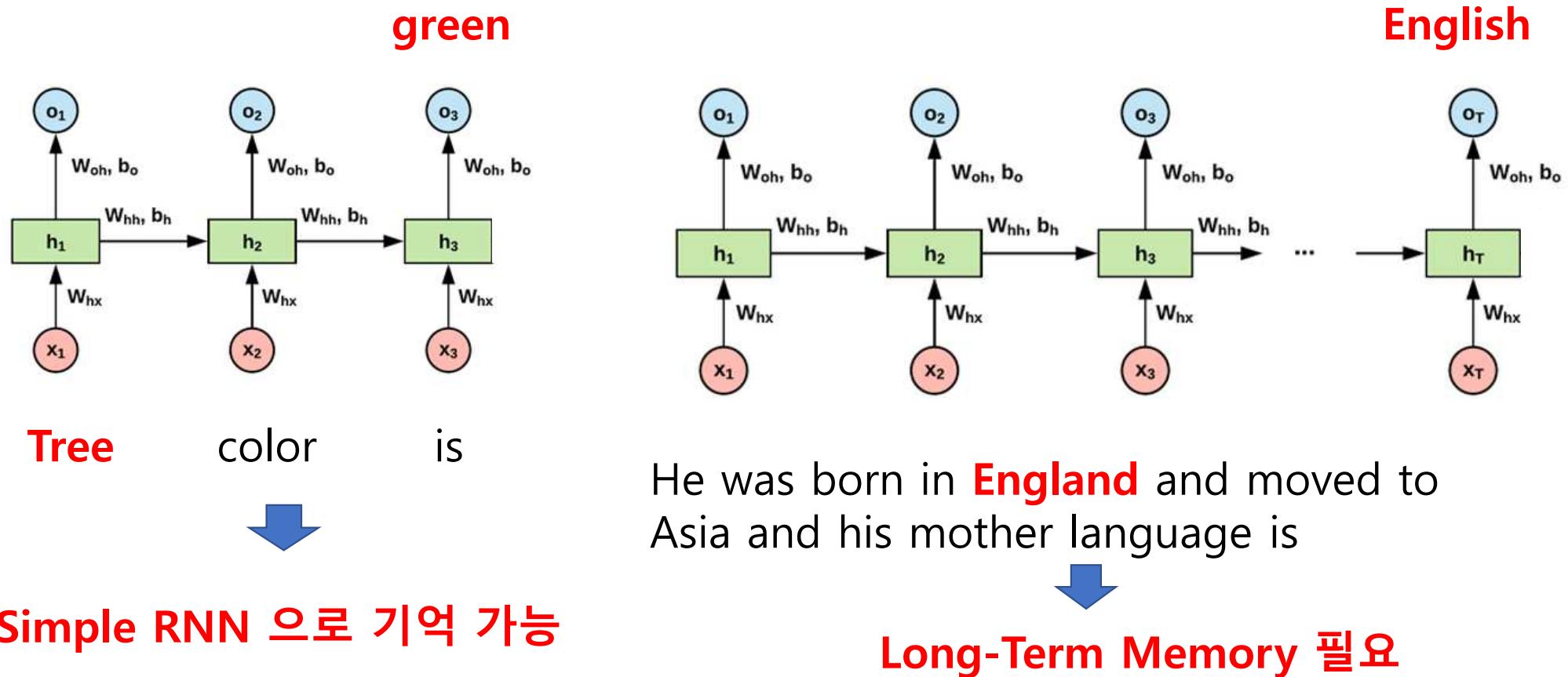
- RNN 을 순서대로 펼쳐 놓으면 weight 를 공유하는 매우 deep 한 neural network 이 된다.
- BPTT (Backpropagation Through Time) 으로 parameter 학습

How RNN is trained ?

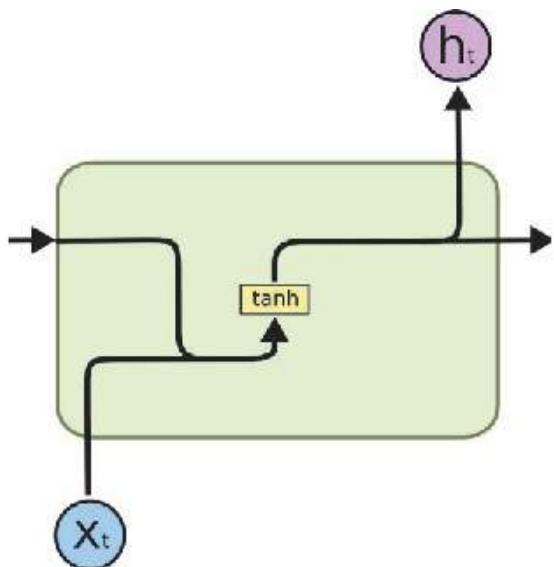


- "h", "e", "l", "o" 4 개 문자만 있다고 가정
- 각 문자를 One-hot encoding
- "h" 를 시작 문자로 주면 "hello" 가 출력 되도록 훈련
- 각 time step 의 target character 는 "hello" 내의 next character
- Gradient descent 와 backpropagation 을 통해 output layer 의 target score 가 증가되도록 함 (green color)
- "l" 다음의 character 는 현재의 "l" 만으로 판단할 수 없음 (history 필요)

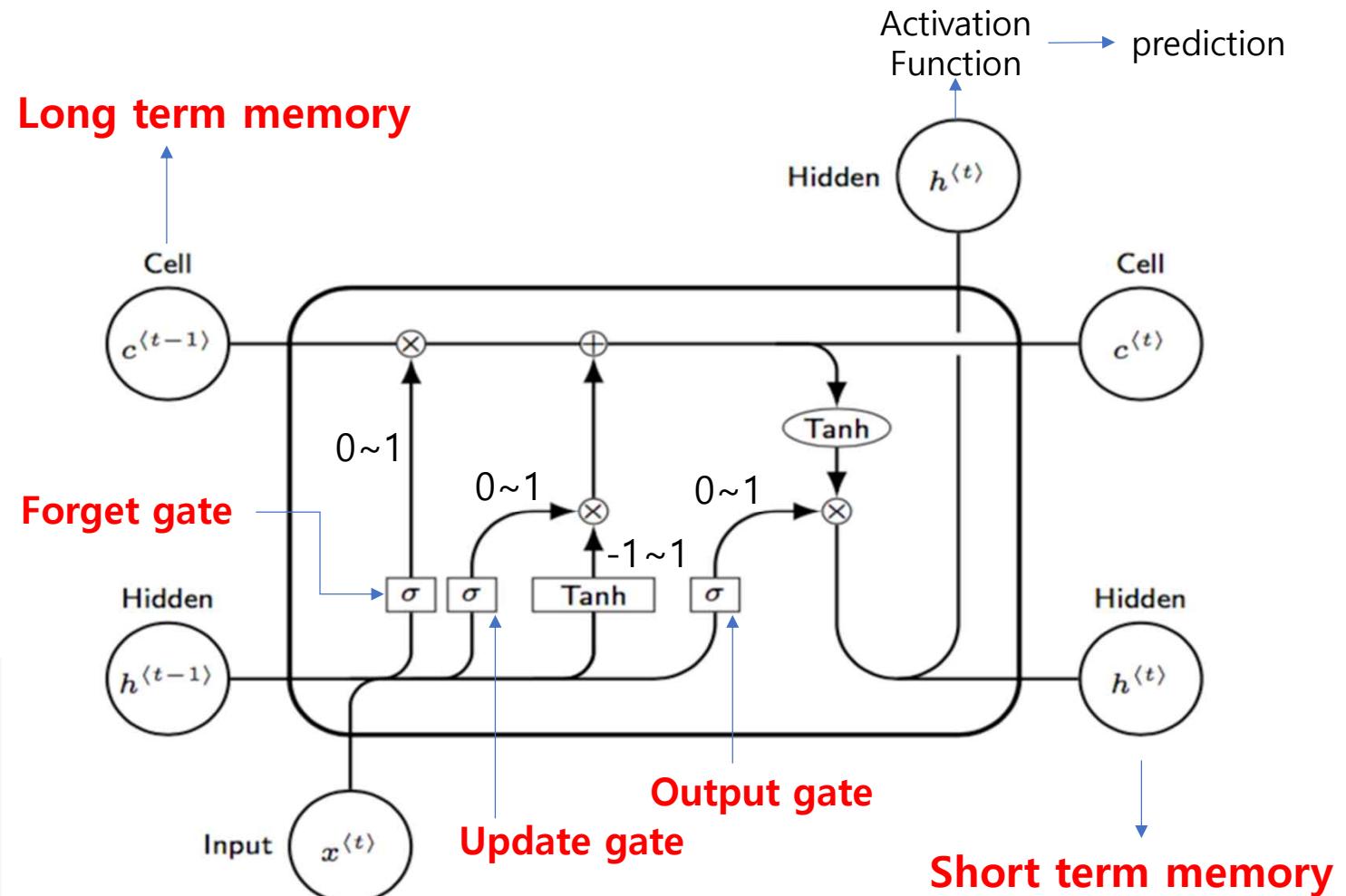
LSTM (Long Short-Term Memory)



SimpleRNN



LSTM (Long Short-Term Memory)



LSTM 내부 구조

- **Input** – 이전 step 의 hidden + new data

$$\tilde{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \rightarrow \text{새로운 cell status 후보}$$

- **Update gate** – input 을 어느정도 받아들일지 결정 (0-무시, 1-전체)

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

- **Forget gate** – 이전 cell state 를 어느정도 기억할지 결정 (0-forget, 1-전체 기억)

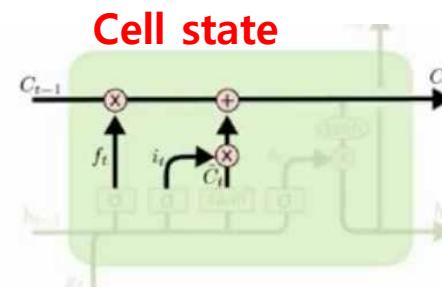
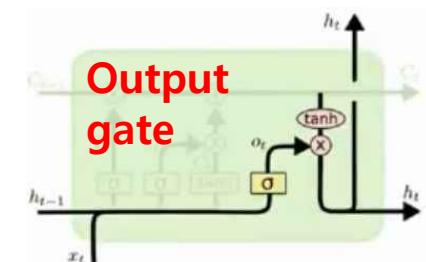
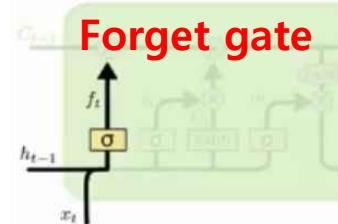
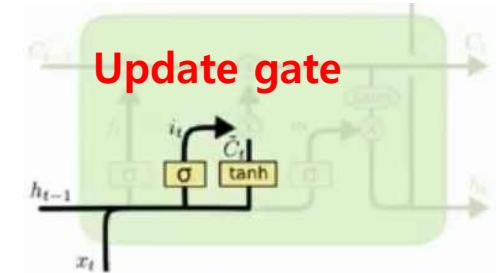
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

- **Output gate** – input 을 어느정도 다음 step 으로 보낼지 결정

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

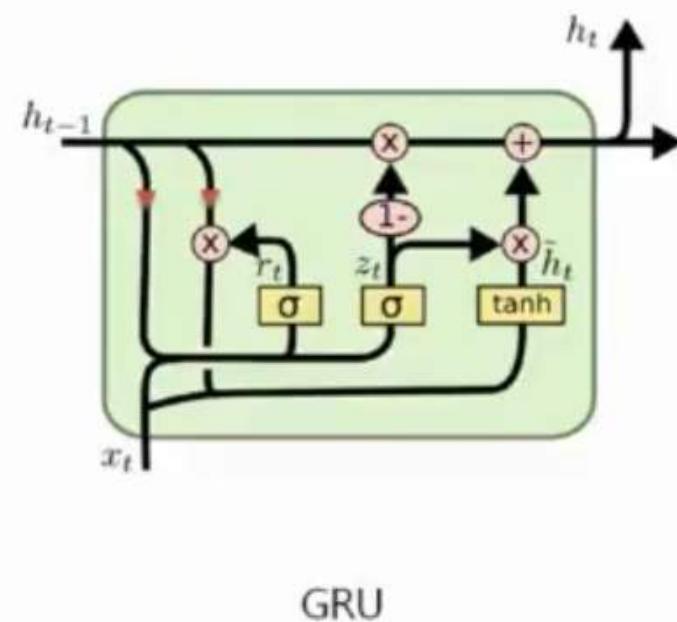
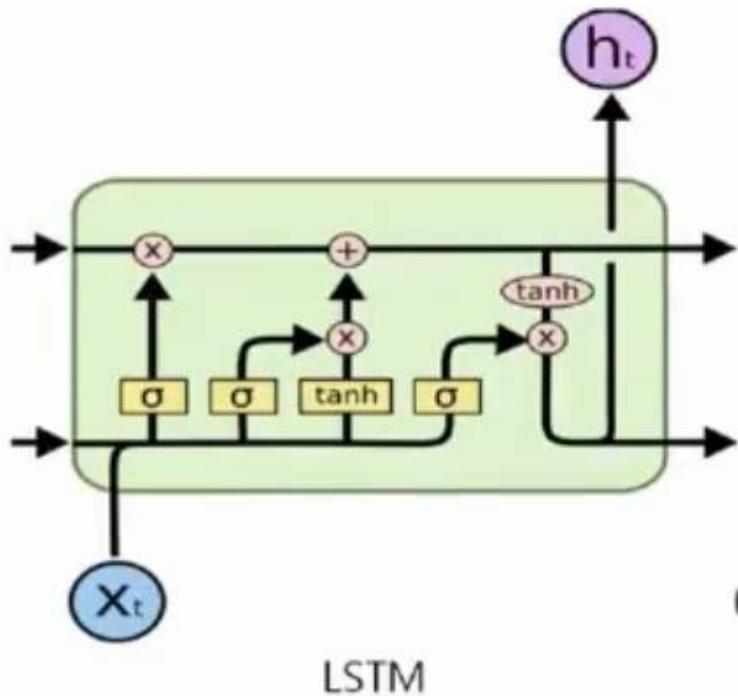
- $C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \Gamma_f * C^{<t-1>}$

- $a^{<t>} = \Gamma_o * \tanh C^{<t>}$



GRU (Gated Recurrent Unit)

- LSTM 의 장점을 유지하면서 일부 Gate 를 생략하여 계산의 복잡성을 낮춤



실습: 020. LSTM/GRU input/output shape

- **LSTM**

- **return_sequences**

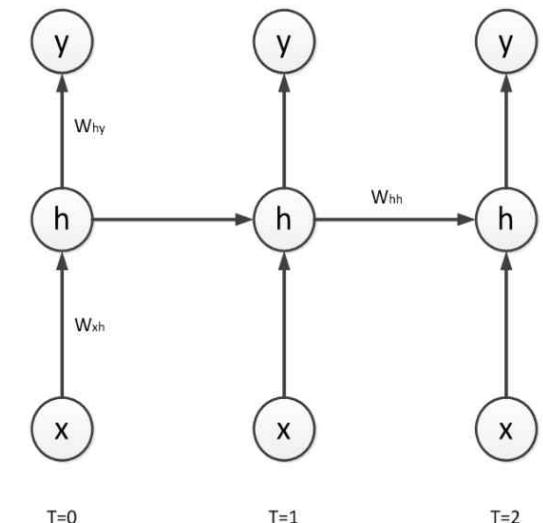
- False (default) - last time step 의 output 만 반환
 - True - 모든 timestep 의 output 을 모두 반환

- **return_state**

- False (default) - output 만 반환
 - True - output, last step 의 hidden state, cell state (LSTM 의 경우) 반환

- **Bidirectional LSTM**

- 순방향, 역방향이 concatenate 된 output 출력
 - hidden state, cell state 는 순방향, 역방향 별도 출력



LSTM – return_sequence, return_state

```
1 from tensorflow.keras.models import Model
2 from tensorflow.keras.layers import Input, LSTM
3 import numpy as np
4
5 T = 4 #Time Steps
6 D = 2 #features
7 U = 3 #LSTM units
8
9 X = np.random.randn(1, T, D)
10 print(X.shape)
11 X
```

(1, 4, 2)

```
array([[[ 1.40105178, -0.45755604],
       [-0.24789063, -1.55088017],
       [-0.91353969, -0.71361525],
       [-1.10306571,  0.76256955]]])
```

```
1 def lstm(return_sequences=False, return_state=False):
2     input_ = Input(shape=(T, D))
3     rnn = LSTM(U, return_state=return_state, return_sequences=return_sequences)
4     x = rnn(input_)
5
6     model = Model(inputs=input_, outputs=x)
7     if return_state:
8         o, h, c = model.predict(X)
9         print("o : ", o, o.shape)
10        print("h : ", h, h.shape)
11        print("c : ", c, c.shape)
12    else:
13        o = model.predict(X)
14        print("o : ", o, o.shape)
15
16    print("---- return_sequences = False ----")
17    lstm(return_sequences=False)
18    print()
19    print("---- return_sequences = True ----")
20    lstm(return_sequences=True)
21    print()
22    print("---- return_state = True ----")
23    lstm(return_state=True)
```

return_sequence=False



---- return_sequences = False ----
o : [[-0.09496244 -0.04834926 0.18206272]] (1, 3)

return_sequence=True



---- return_sequences = True ----
o : [[[0.08876073 0.07523488 -0.10073236]
 [-0.37540612 0.2561414 -0.04929614]
 [-0.2844468 0.29576355 -0.1451881]
 [-0.14998941 0.20151682 -0.1950384]]] (1, 4, 3)

return_state=False



---- return_state = True ----
o : [[-0.08381992 0.00473488 -0.07198517]] (1, 3)
h : [[-0.08381992 0.00473488 -0.07198517]] (1, 3)
c : [[-0.15797216 0.01051851 -0.1415057]] (1, 3)

return_sequence=False



return_state=True

Bi-directional LSTM – [순방향, 역방향]

```
1 def lstm(return_sequences=False, return_state=False):
2     input_ = Input(shape=(T, D))
3     rnn = Bidirectional(LSTM(U, return_state=return_state, return_sequences=return_sequences))
4     x = rnn(input_)
5
6     model = Model(inputs=input_, outputs=x)
7     if return_state:
8         o, h1, c1, h2, c2 = model.predict(X)
9         print("o :", o, o.shape)
10        print("h1 :", h1, h1.shape)
11        print("c1 :", c1, c1.shape)
12        print("h2 :", h2, h2.shape)
13        print("c2 :", c2, c2.shape)
14    else:
15        o = model.predict(X)
16        print("o :", o, o.shape)
17
18    print("---- return_sequences = False ----")
19    lstm(return_sequences=False)
20    print()
21    print("---- return_sequences = True ----")
22    lstm(return_sequences=True)
23    print()
24    print("---- return_state = True ----")
25    lstm(return_state=True)
```

- Output :
timestep T forward state + timestep 1 backward state

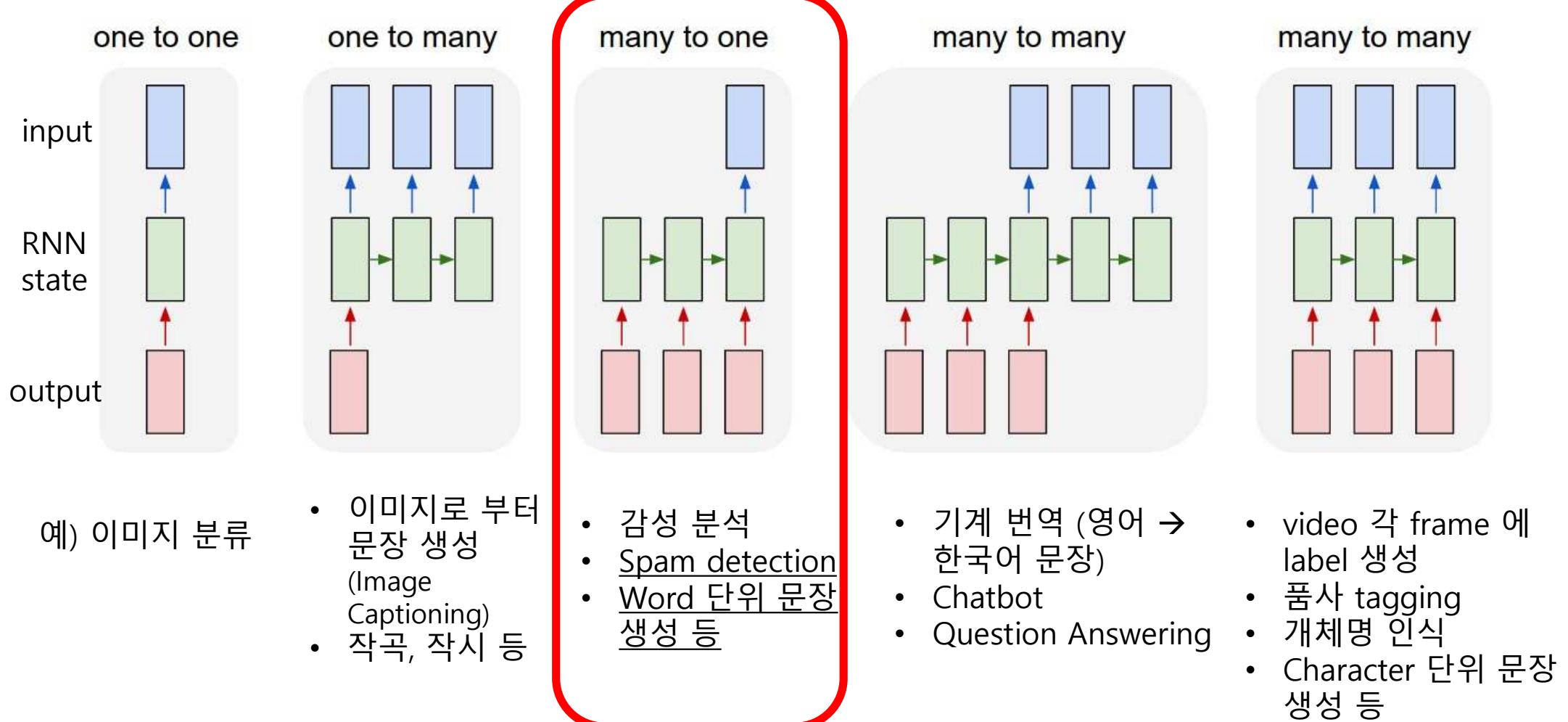
---- return_sequences = False ----
o : [[-0.05495989 -0.10378188 0.19712731 0.1560666 -0.07385919 -0.15323249]] (1, 6)

---- return_sequences = True ----
o : [[[-0.0676645 0.0364071 0.28644037 0.18609144 0.57841843
-0.34381726]]
[0.07355319 0.02145659 0.02598272 -0.02505665 0.12553862
-0.10975165]]
[-0.02330432 0.01271209 0.31423712 0.21903777 0.39983
-0.27724865]]
[-0.13461797 0.17682895 0.13418674 0.00944902 0.05306194
-0.07824305]]] (1, 4, 6)

---- return_state = True ----
o : [[0.13395719 0.05881313 -0.18529123 -0.15092295 0.49242502 0.24727458]] (1, 6)
h1 : [[0.13395719 0.05881313 -0.18529123]] (1, 3)
c1 : [[0.27381814 0.14079721 -0.56539357]] (1, 3)
h2 : [[-0.15092295 0.49242502 0.24727458]] (1, 3)
c2 : [[-0.2220205 0.67217296 0.40521646]] (1, 3)

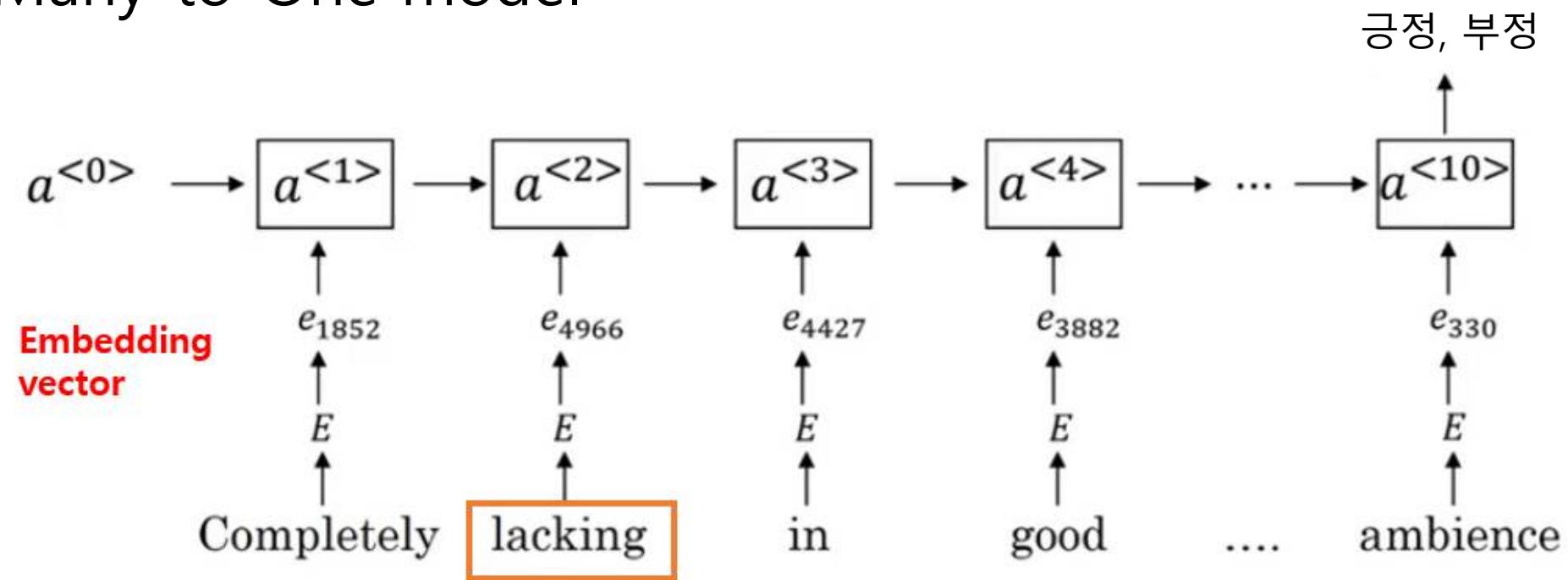
Sentiment Analysis (감성분석)

RNN 을 이용한 Sentiment Analysis



RNN for sentiment classification

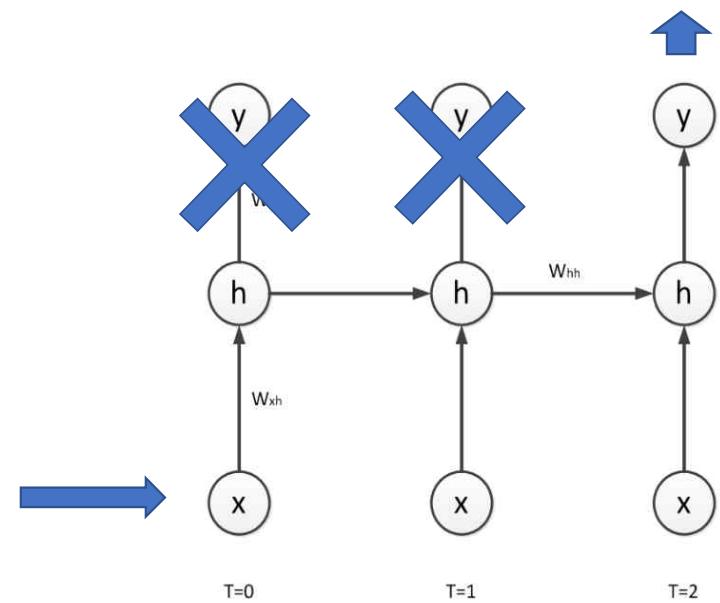
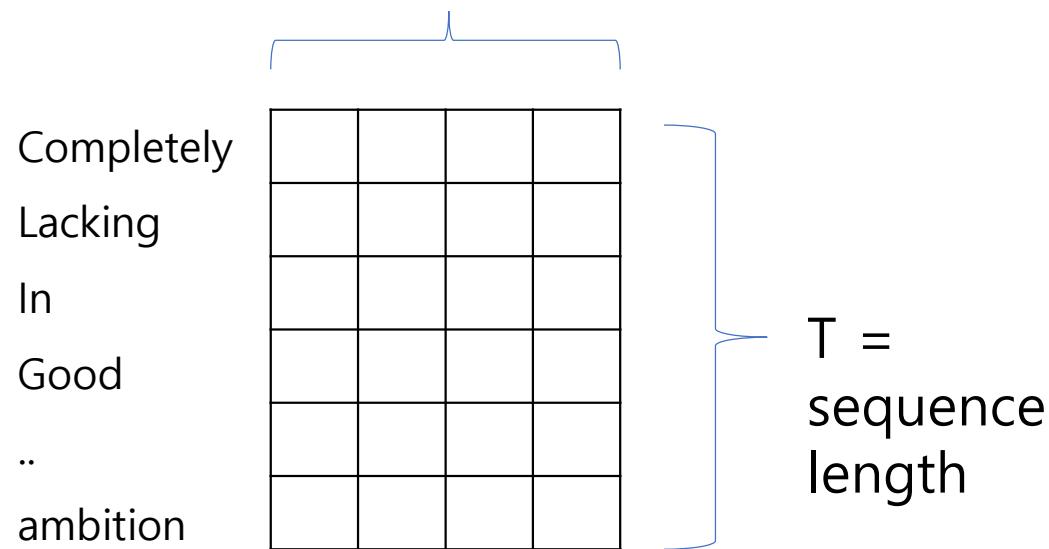
Many-to-One model



RNN input / output

Sentiment analysis 는 last y 만 필요
return_sequences = False

D = embedding dimensionality



실습 : 030-IMDB 영화 관람평 분류

- Deep Learning NLP 의 hello world
- tensorflow tfds dataset 의 “imdb_reviews” 를 이용
 - 각 25,000 개의 training/testing set 으로 구성된 IMDB 영화관람평
- “imdb_reviews” – encoding 되어있지 않은 string 형태의 data
- “imdb_reviews/subwords8k”
 - 8,000 개의 subwords 로 encoding 되어 있는 sequence data
- Movie reviews sentiment classification
 - Label : positive, negative → binary classification

실습 : 035-네이버 영화 관람평 분류

- 150,000 개의 training set, 50,000 개의 testing set 으로 구성
- Label : positive, negative → binary classification
- 처리 방법은 영어 데이터에 대한 텍스트 분류와 기본적으로 동일
- 한국어 데이터는 토큰화(tokenization)를 할 때 형태소 분석기를 사용한다는 차이점
- null data 가 있으므로 data cleansing 필요
 - Colab 에서 약 8 분 소요

- 한글이외의 character 삭제 및 불용어 처리 등 전처리 필요
 - 띄어쓰기가 잘 지켜지지 않으므로 한글 형태소 분석기 필요
-
- from konlpy.tag import Okt
okt = Okt()
okt.morphs("열심히코딩한당신, 아버지가방에들어가신다")
['열심히', '코딩', '한', '당신', ',', '아버지', '가방', '에', '들어가신다']

Embedding layer 시작화

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(VOCAB_SIZE+1, EMBEDDING_DIM, input_length=40),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64, dropout=0.2, return_sequences=True)),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, dropout=0.2)),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

```
1 e = model.layers[0]
2 weights = e.get_weights()[0]
3 print(weights.shape)          # shape: (vocab_size, embedding_dim)
4 print(weights[0])
```

```
(10000, 16)
[-0.01035169  0.02202568  0.00211992 -0.0348761 -0.0072158 -0.00667641
 0.00071818 -0.04750429 -0.03458726 -0.06299702  0.04483034 -0.04684689
 -0.01423898 -0.05427958  0.01500437  0.02816296]
```

- Embedding 은 **10000** words from corpus X **16** dimension features 로 구성
- Embedding projector <https://projector.tensorflow.org/> 를 이용하여 시각화

3D plotting 용 embedding file 저장

- vector file & meta data file

```
import io

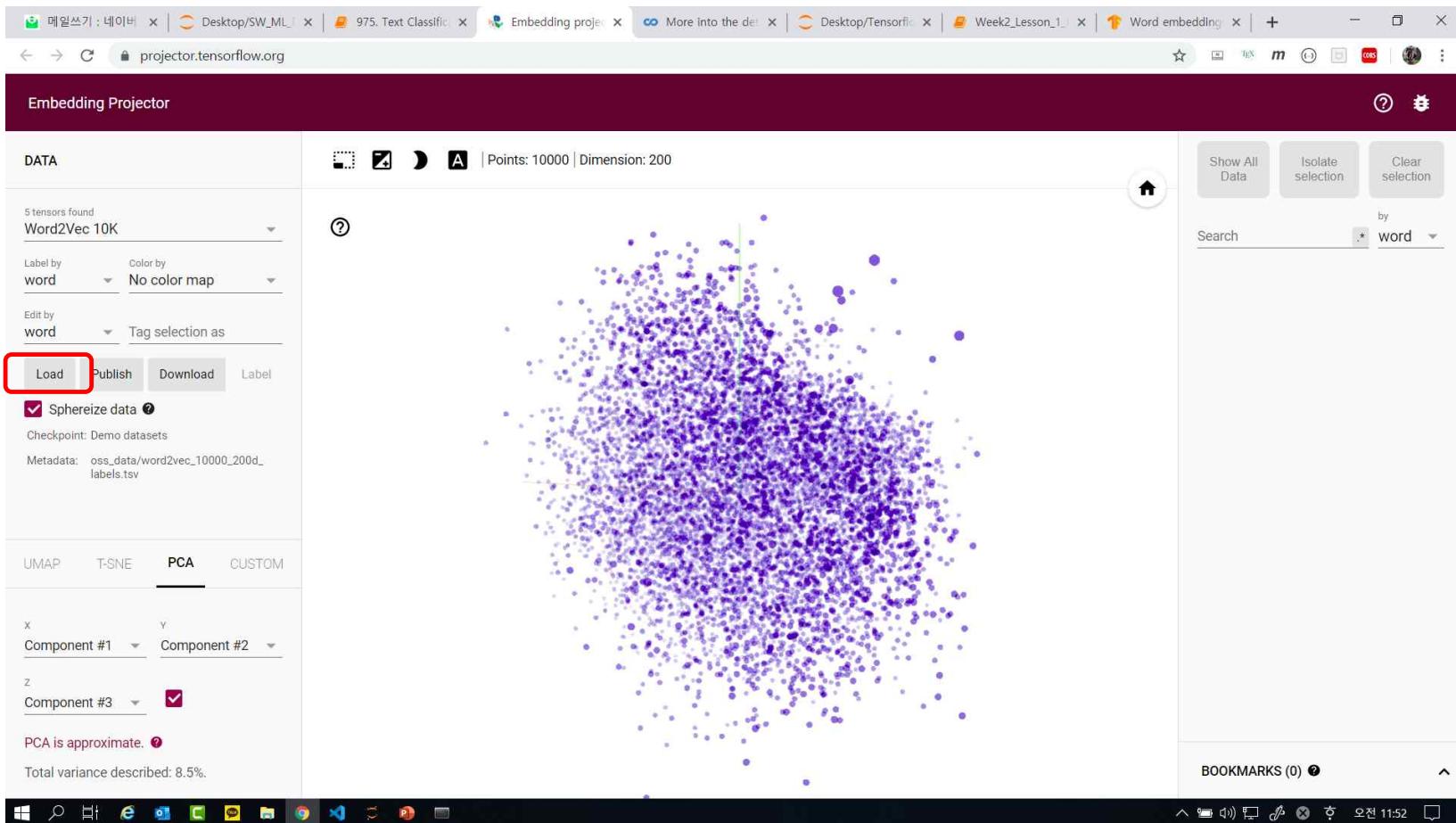
out_v = io.open('./data/vecs.tsv', 'w', encoding='utf-8')
out_m = io.open('./data/meta.tsv', 'w', encoding='utf-8')

for i in range(1, 1000):
    word = tokenizer.index_word.get(i, '?')
    embeddings = weights[i]
    out_m.write(word + "\n")
    out_v.write('\t'.join([str(x) for x in embeddings]) + "\n")

out_v.close()      Embedding 값을 disk에 저장
out_m.close()
```

Embedding projector

<https://projector.tensorflow.org/>



vector, meta file upload

Load data from your computer

Step 1: Load a TSV file of vectors.

Example of 3 vectors with dimension 4:

```
0.1\t0.2\t0.5\t0.9  
0.2\t0.1\t5.0\t0.2  
0.4\t0.1\t7.0\t0.8
```

vecs.tsv

Choose file

Step 2 (optional): Load a TSV file of metadata.

Example of 3 data points and 2 columns.

Note: If there is more than one column, the first row will be parsed as column labels.

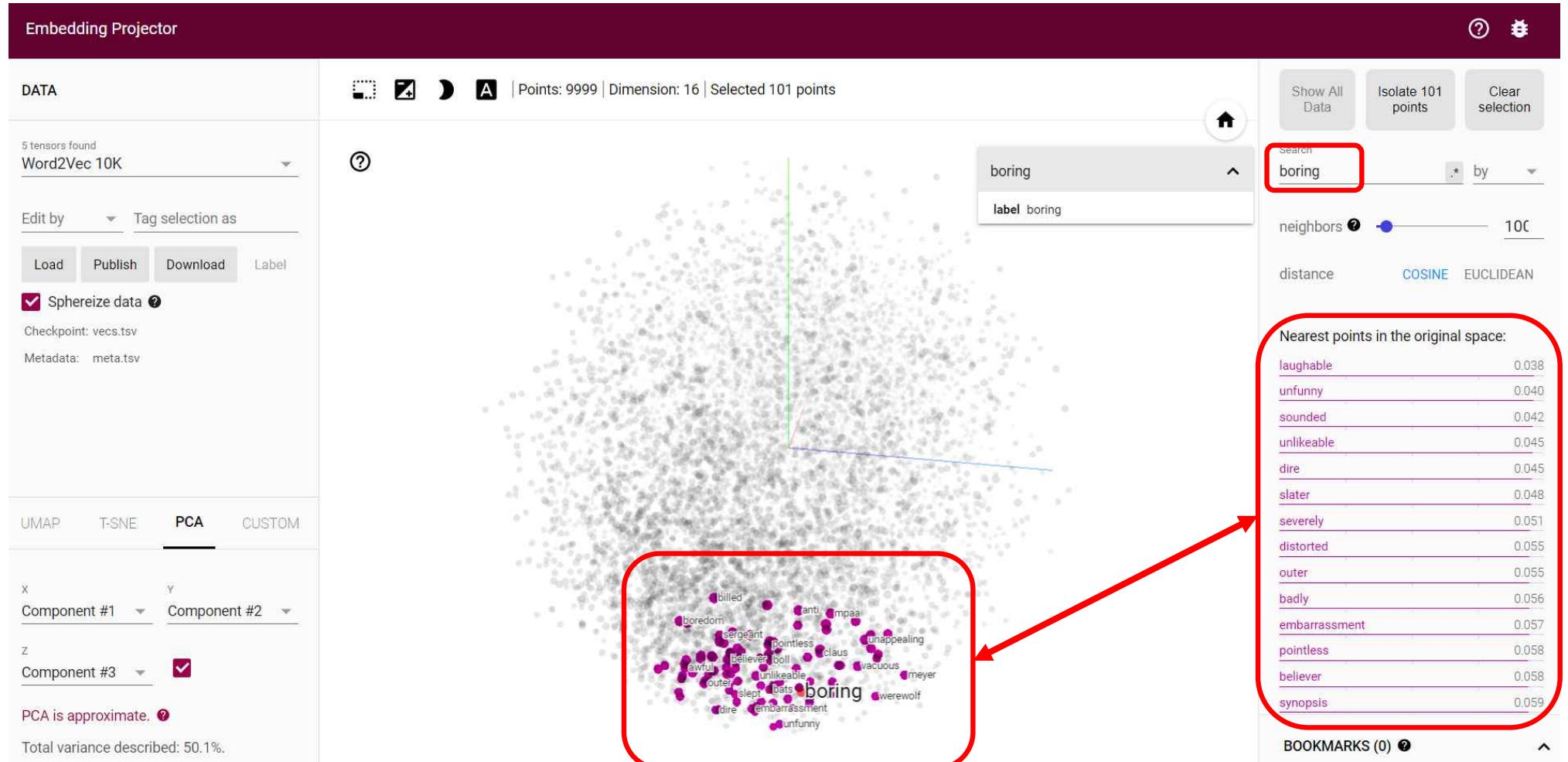
```
Pokémon\tSpecies  
Wartortle\tTurtle  
Venusaur\tSeed  
Charmeleon\tFlame
```

meta.tsv

Choose file

Click outside to dismiss.

- ✓ Colab 사용시 local로 download 후 projector에 load



Tokenization & Tokenizer

- Tokenization

- text 를 word 또는 sub-word 로 분리하는 것

(코, 명사),
(코로나, 명사),
(로, 조사)
(나가, 동사)
(나, 조사)
...

단어/형태소 사전

- 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

코, 로, 나, 가, 심, 하, 다,
코로, 코로나, 로나, ...,
가</w>, 다</w>,
...

서브워드 사전

Tokenizer 비교

Tokenizer	토큰 단위	vocab size	미등록단어에 대한 가정
사전 기반	알려진 단어/형태소 및 이들의 결합	unlimited	<ul style="list-style-type: none">알려진 단어/형태소의 결합이라 가정필요시 형태소 분석 (형태 변형 가능)빈번하지 않은 단어를 사전에 등록해 두면 잘 인식사전에 등록되지 않은 단어는 UNK 처리
sub-word	알려진 글자 및 subword	fixed	<ul style="list-style-type: none">알려진 sub-words로 분해 ex) appear → app + ear자주 등장하는 단어를 제대로 인식할 가능성 높음알려진 글자로 분해하여 UNK의 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 token화에 유용
Bert 104 개국어 version 은 110,000 vocabulary size

WPM(Word Piece Model) 개요

- 하나의 단어를 내부단어(subword)로 통계에 기반하여 띄어쓰기로 분리.
- 2015년 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(characters) 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 Source로 개방하여 실무에 사용 가능 (**한국어에 적용 가능**)
- SentencePiece는 Unigram 알고리즘 사용

실습 : 037-Tokenizer Train 실습

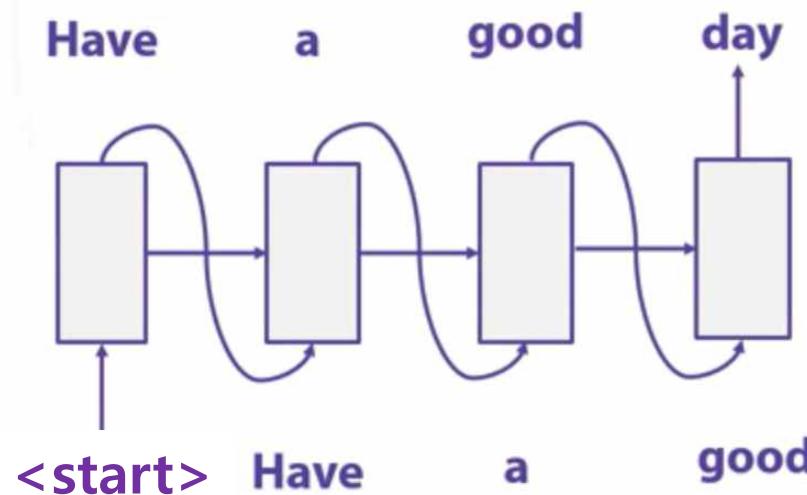
- Keras tokenizer
- KoNLPy Okt tokenizer
- Google Sentencepiece
 - pip install sentencepiece
 - NSMC (Naver Sentiment Movie Corpus) data 를 이용한 tokenizer train
 - 아 더빙.. 진짜 짜증나네요 목소리
 - ['_아', '_더빙', '..', '_진짜', '_짜증나네요', '_목소리']
 - [52, 752, 5, 25, 16020, 1401]

Language Model

language model – next word prediction model

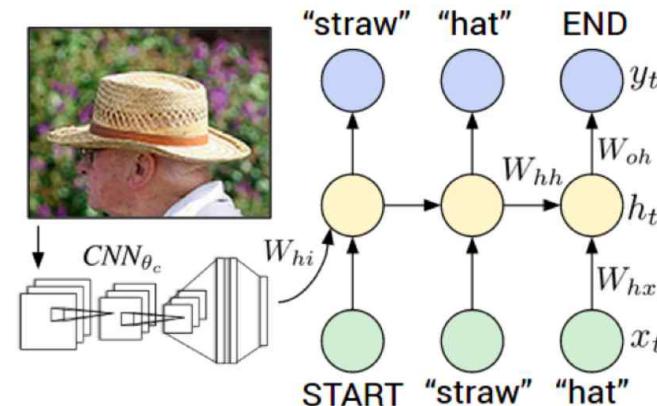
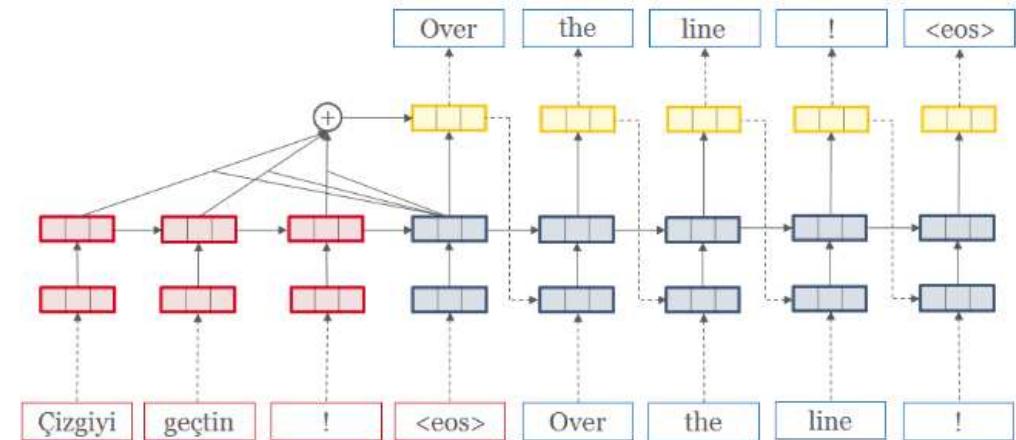
How to generate language ?

- 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



적용 분야

- Machine Translation
- Question Answering
- Chatbot
- Speech Recognition
- Text Summarization
- Image Captioning
- 기타



Seq2Seq

Language Translation

(기계번역)

Machine Translation Task 활용 범위

- Language에 상관 없이 동일한 model 적용 가능
 - **Encoder-decoder model**은 활용도가 매우 높은 중요한 기술
- 질의 응답 문제 (story + question → answer) 에도 적용 가능
 - **story + question** : input sequence → thought vector encoding
 - **answer** : output sequence → thought vector decoding

ex) Story + Question :

 아인슈타인의 Wikipedia page + “아인슈타인은 어떤 이론으로 유명한가 ?”
 A : “상대성이론”

BLEU (Bilingual Evaluation Understudy)

- 기계번역의 품질을 평가하는 알고리즘
- 언어에 무관하며 이해하기 쉽고, 계산하기 쉽다
- [0, 1] 사이의 값. Higher score, Better quality.
- Score 계산법 – unigram or bigram
references : "the weather is extremely good"
(the, weather), (weather, is), (is, extremely), (extremely, good)
predicted result = "the weather is good"
(the, weather), (weather, is), (is, good)

$$\text{BLEU} = \frac{1}{3} + \frac{1}{3} + \frac{0}{3} = 0.666$$



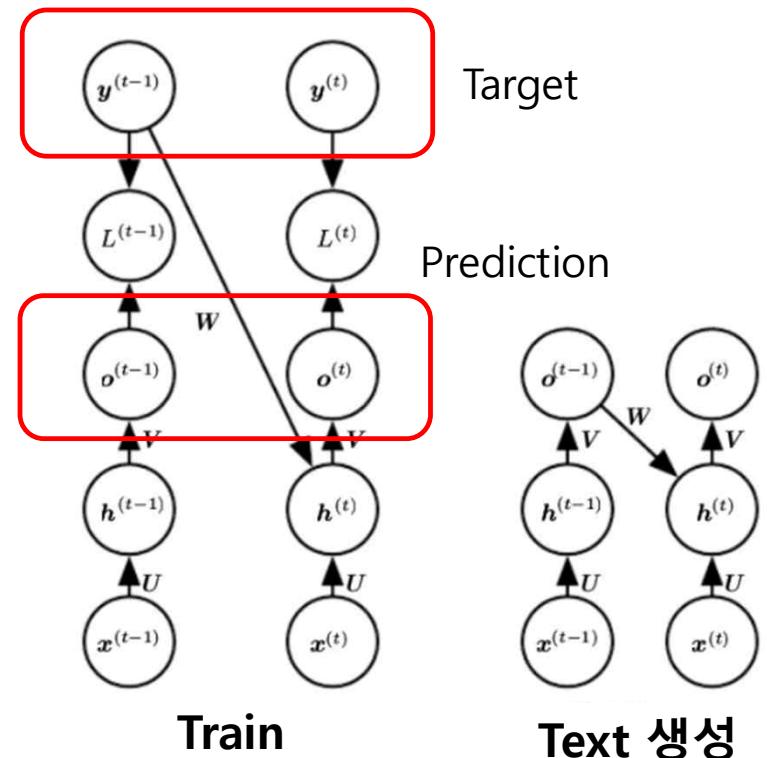
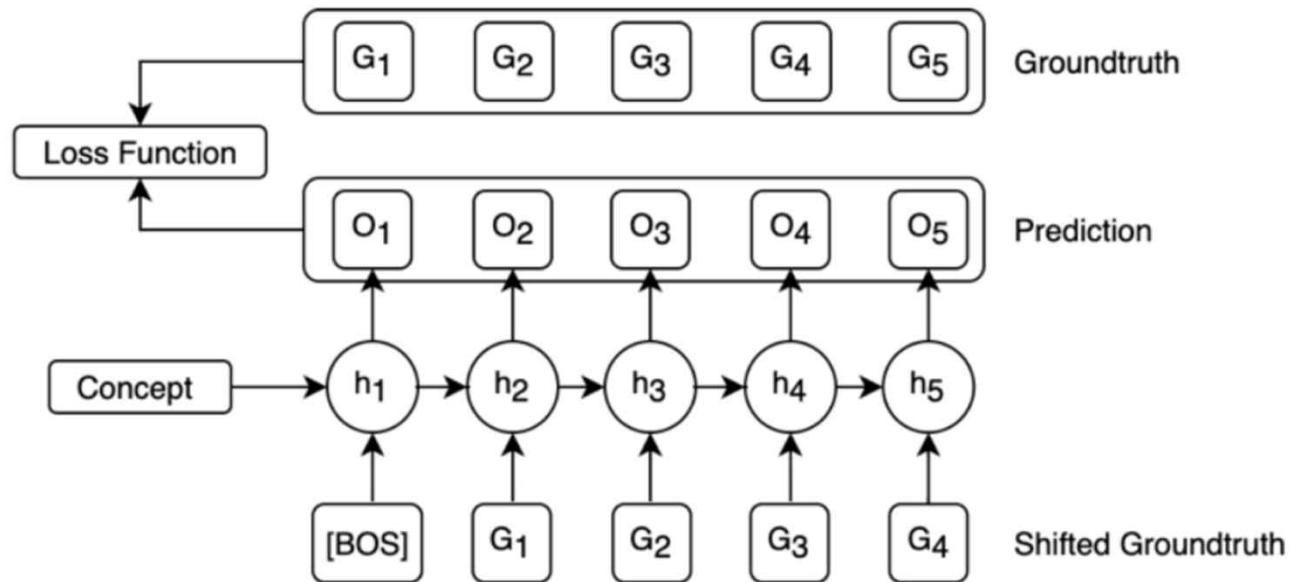
(the, weather) (weather, is) (is, good)

What is Teacher Forcing ?

- 첫번째 단어에서 잘못된 예측을 했을 경우 시간이 지날수록 더 큰 잘못된 예측을 할 가능성 증가
- 이에 따라 문장(시퀀스)의 구성이 바뀔 뿐만 아니라, 예측 문장 (시퀀스)의 길이 마저도 바뀜.
- 학습 과정에서는 이미 정답을 알고 있고, 현재 모델의 예측 값과 정답과의 차이를 통해 학습하므로, 실제 값을 다음 단어 예측의 입력 값으로 사용
- 교사가 학생에게 번역을 가르치는 방법과 유사 (단계별 수정)

Teacher Forcing

- 이전 timestep 의 output 을 input 으로 사용
- 한번에 전체 sentence 를 맞추는 것은 힘드므로 단어 단위로 교사가 교정해 주듯 전체 문장 완성



Teacher Forcing

model 2

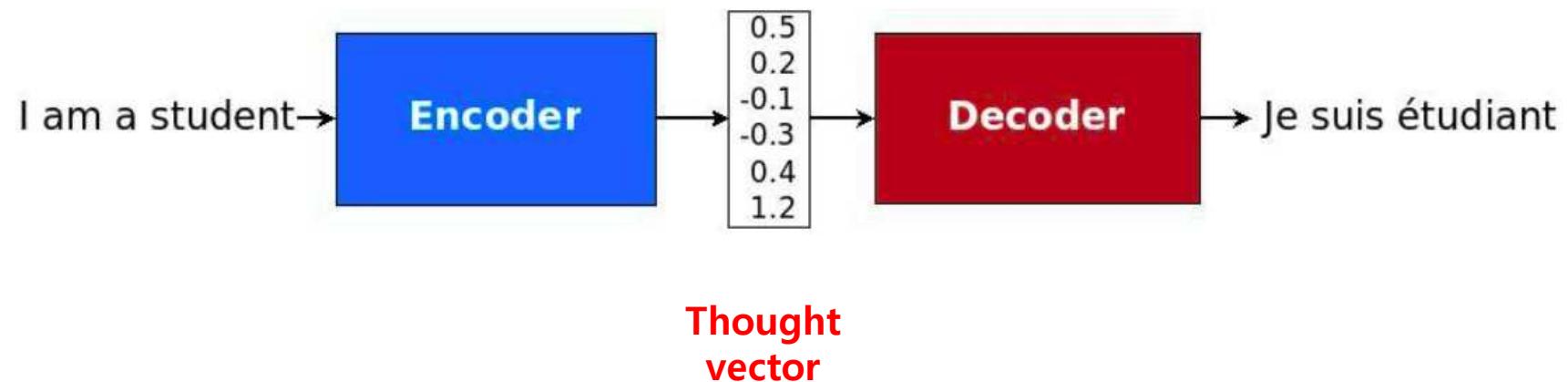
Encoder-Decoder Model

전통적 기계번역 vs NMT(Neural Machine Translation)

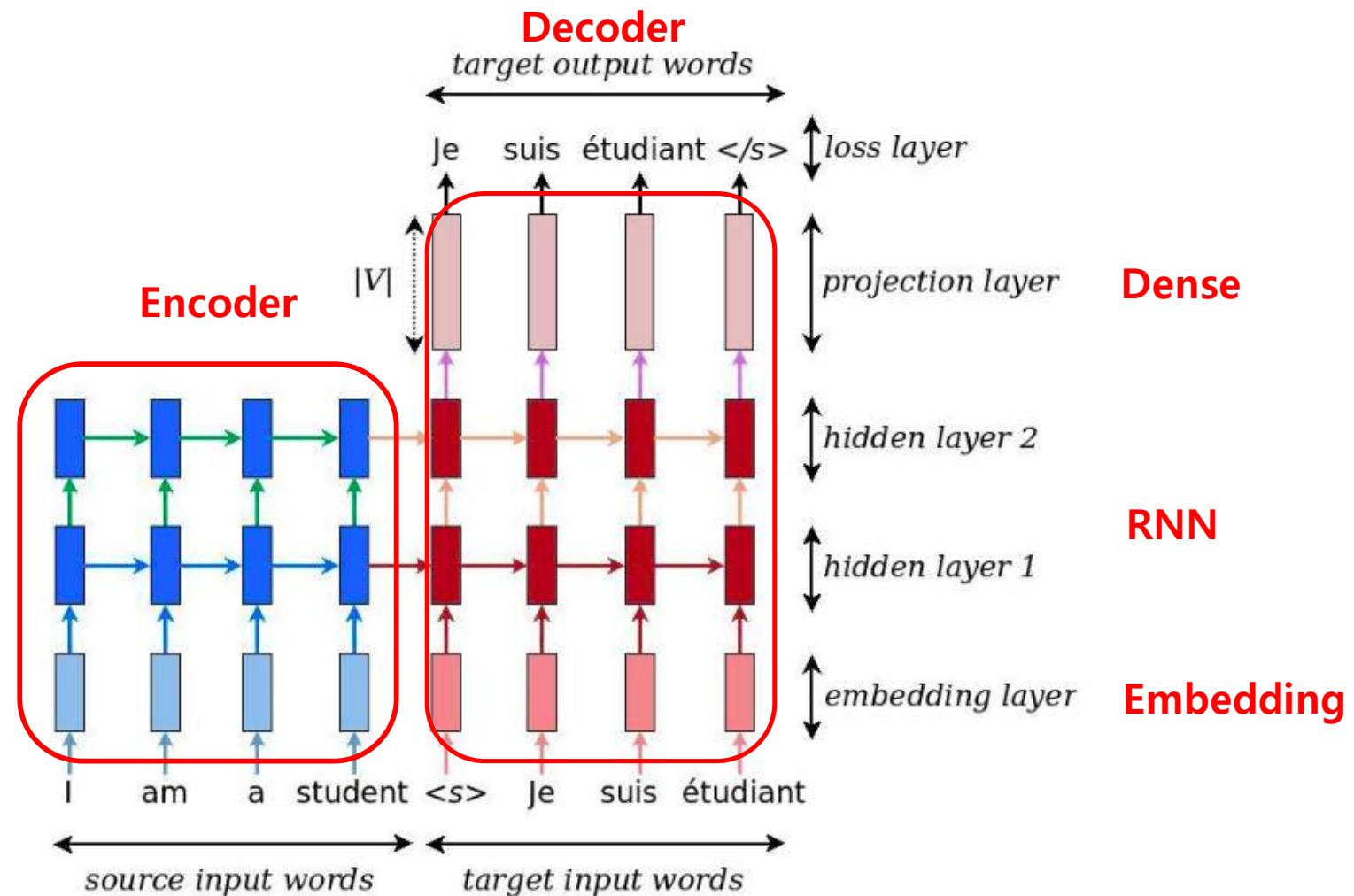
- 전통적 Machine Translation → 구문단위로 분리하여 번역



- Encoder-Decoder model
→ 문장 전체를 읽고 의미를 이해한 다음 번역, 인간의 방식과 유사

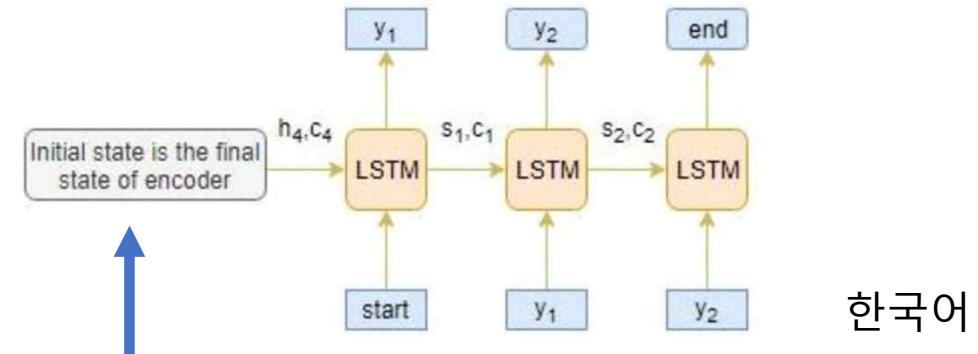
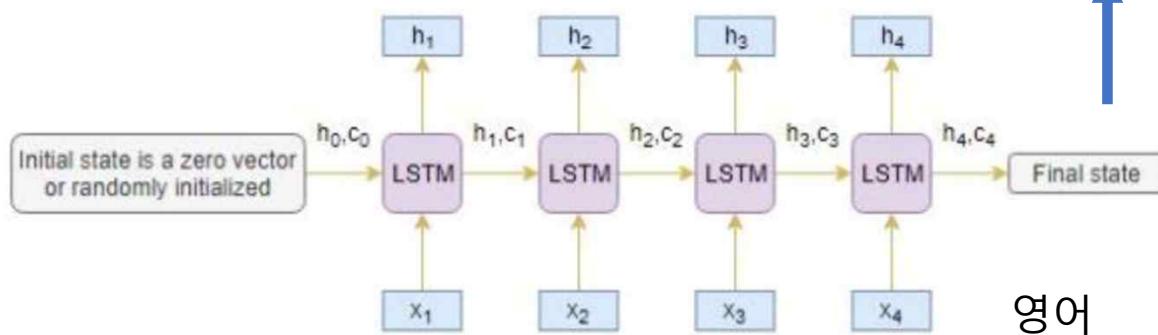


Encoder-Decoder Model 의 Network 구조



Training Phase

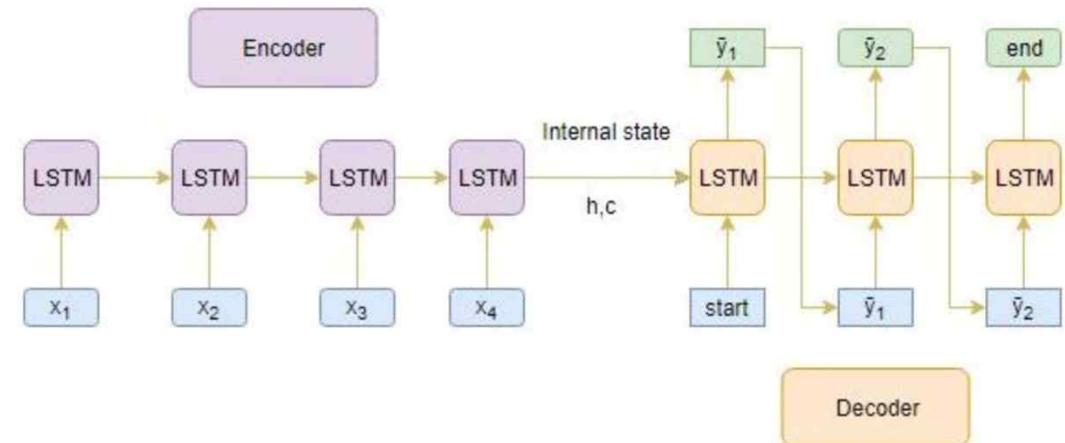
- Teacher Forcing 기법 사용



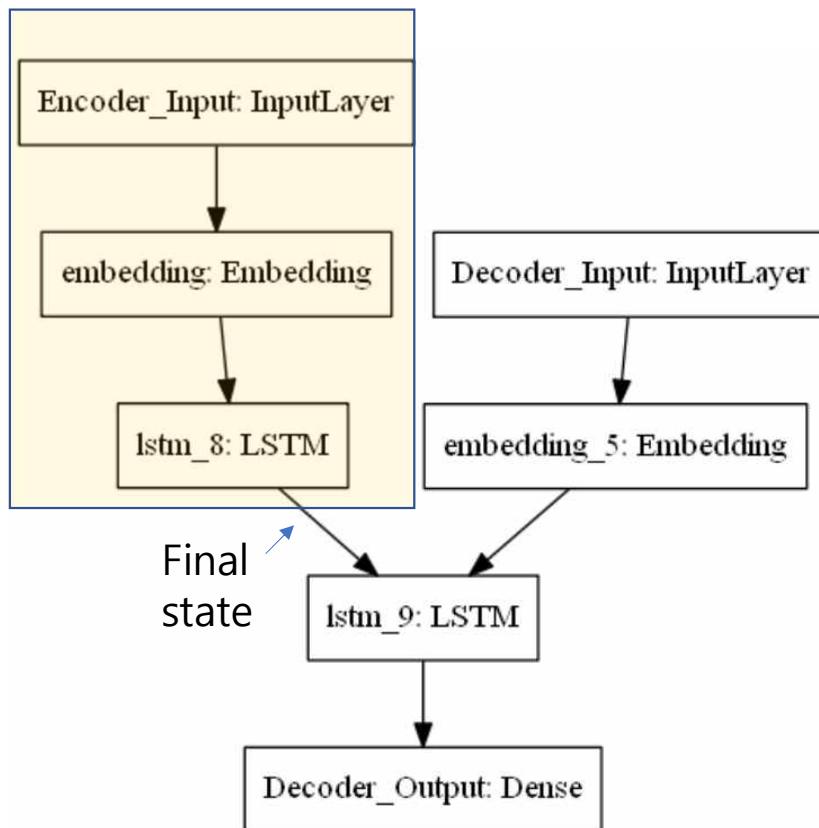
Teacher Forcing

Inference Phase

- Text Generation 기법 사용

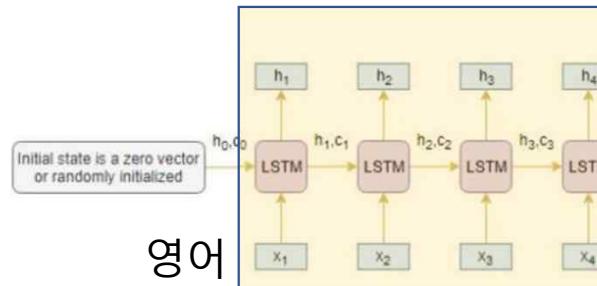


Training Step 의 Model : Encoder + Decoder

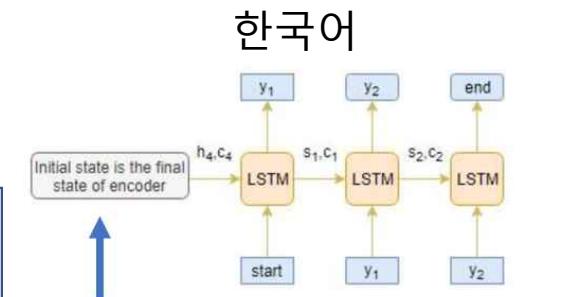


Training Phase

– Teacher Forcing 기법 사용



영어

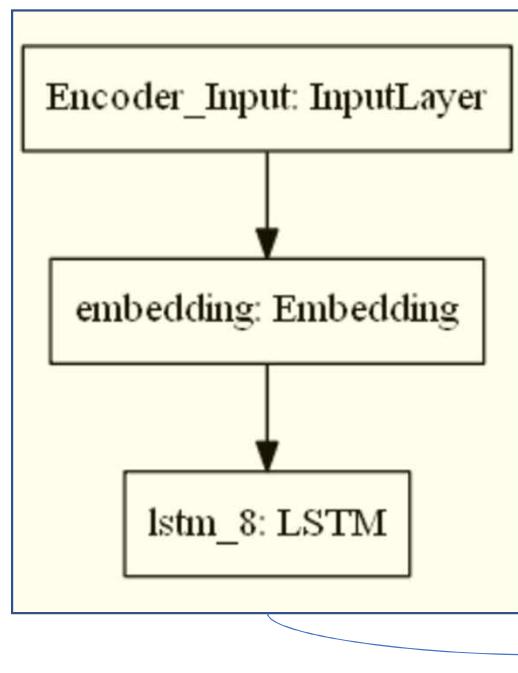


한국어
Teacher Forcing

• Teacher Forcing

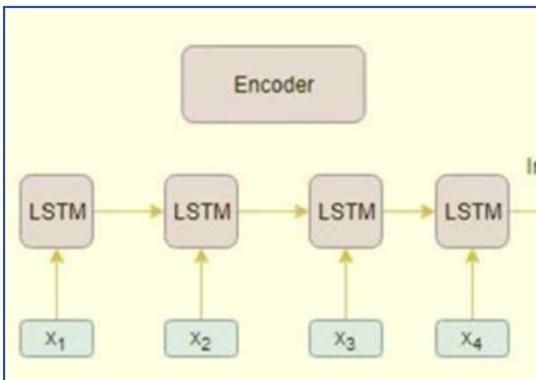
Prediction step 의 model : Encoder, Decoder 분리

영어 sequence



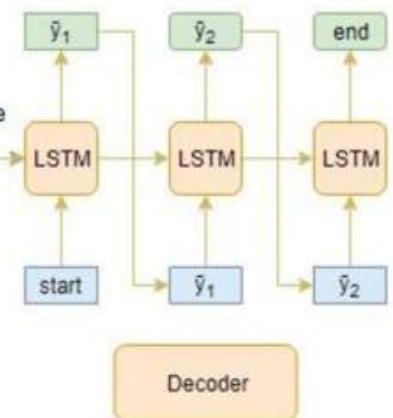
Input_sequence: InputLayer

embedding_5: Embedding



Inference Phase

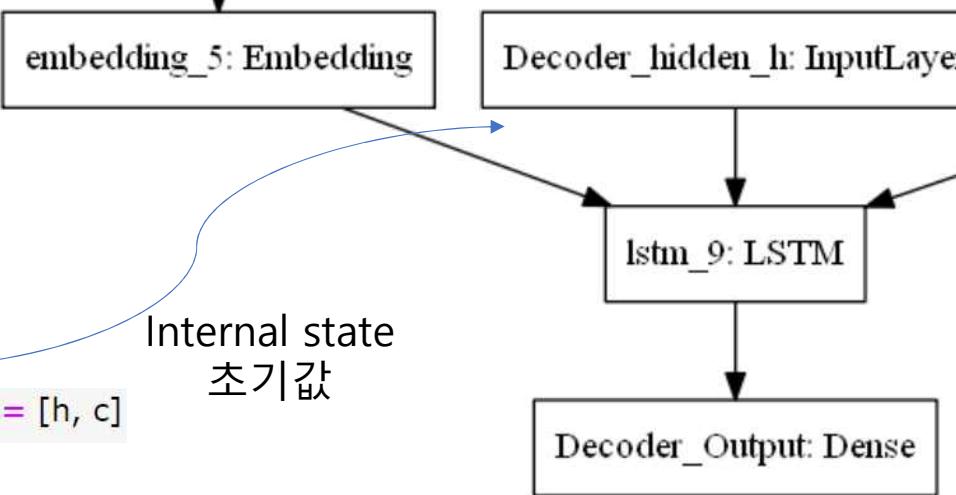
– Text Generation 기법 사용



Decoder

Internal state
초기값

`encoder_states = [h, c]`



Decoding Strategy

- Greedy 전략
 - softmax 분포 중 가장 높은 확률 (argmax) 선택, 언제나 같은 번역
 - argmax $P(\text{Over} \quad \text{the} \quad \text{line} \quad ! \mid \text{Çizgiyi} \quad \text{geçtin} \quad !)$
- Sampling 전략 (사후확률분포)
 - 분포 확률에 따라 Random sampling, 매번 번역이 바뀔 수 있음
 - `np.random.choice(len(probs), p=probs)`
- Beam-search 전략
 - 단순히 첫번째 단어를 argmax로 선택하면 1스텝에서라도 문법 상 실수를 할 경우, 전체 문장의 번역에 큰 실수가 되므로, 각각의 타임스텝 t 마다 b 개의 sequence 후보군을 유지

Highest joint probability

Beam Search Decoding



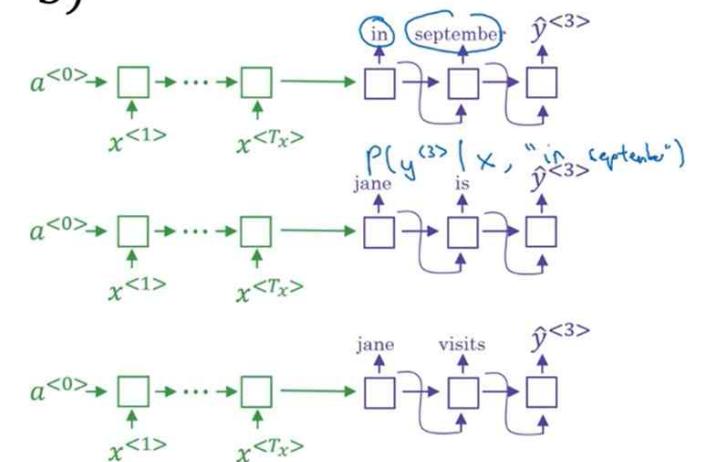
Beam search ($B = 3$)

in september

jane is

jane visits

$$P(y^{<1>} , y^{<2>} | x)$$



jane visits africa in september. <EOS>

Andrew Ng

실습 : 080-seq2seq 기계번역 model 작성

- <http://www.manythings.org/anki/> 사용
 - Kor-eng.zip (English-Korea 번역) → 3798 문장 as of 2022
 - 다른 언어 대비 Corpus 가 적어 품질 떨어지나 학습용 사용 가능
 - Chinese (Mandarin) - English [cmn-eng.zip](#) (21621)
 - French - English [fra-eng.zip](#) (174481)
 - Japanese - English [jpn-eng.zip](#) (45815)
- Tokenization
 - language 가 2 개 이므로 서로 다른 tokenizer 생성. 따라서, 2 개의 word_index 필요
- encoder 와 decoder 의 Embedding layer 에 pre-trained embedding weight 를 초기값으로 load

Transformers

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

NLP Mountain

ChatGPT/Gemini/Lama

2022

BERT/GPT/T5

"Attention is all you need"

Transformer

beginner → Can Start from here ?

2017

Transformer
이해에 필요한
사전 지식

Attention

Encoder-Decoder

Bi-LSTM

RNN

LSTM

2014

NLTK, Konlpy, BOW, TF-IDF, HMM(Viterbi), CNN

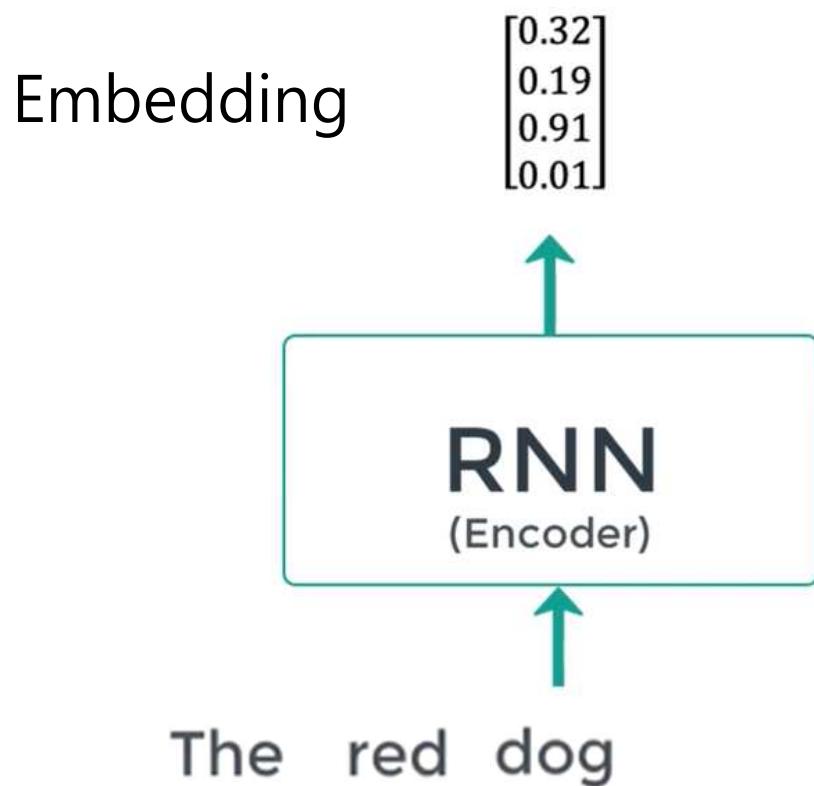
LSTM을 이용한 NLP network의 한계

- Very Long Gradient Path
 - 100 개 단어로 구성된 문장의 LSTM 처리는 100-layer network 깊이에 해당 하므로 vanishing gradient 문제가 여전히 발생
- Transfer Learning 을 하기 어려움
- 특정 Task 마다 별도로 label 된 dataset 이 필요
- RNN 은 time step 순으로 **순차 처리** 하므로 GPU 를 fully 활용 못함

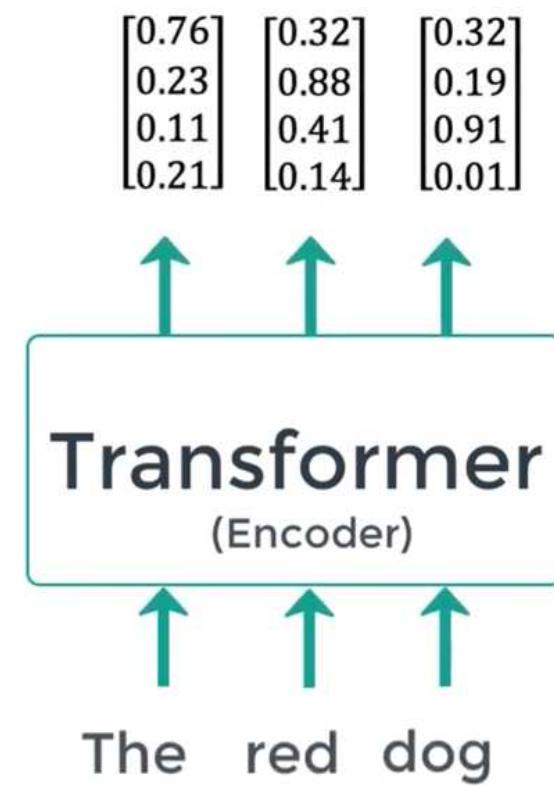
Transformer Model 의 특징

- RNN 혹은 CNN 을 이용하여 sequence-align 하지않은 최초의 **self-attention** model → **"Attention is all you need"**
- 기존의 **encoder-decoder 구조**를 유지
- RNN 을 제거함으로 **병렬처리** 가능 (GPU 효율 극대화)
- 1.6GB 의 WMT2014 English-German Translation task dataset 과 P100 GPU 8 개를 이용하여 3.5 일 train → BLEU score SOTA 달성

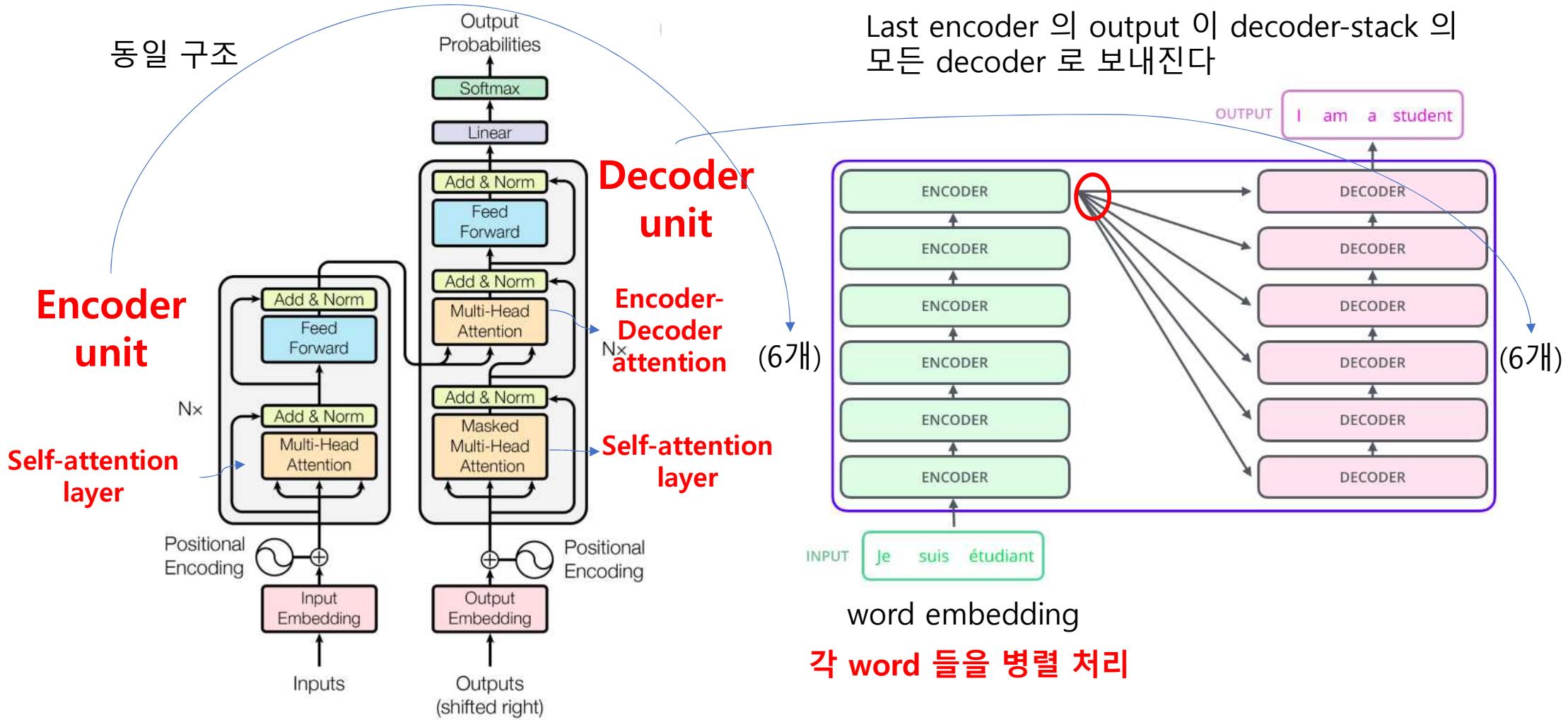
Word Embedding 은
time step 별로 순차적 생성



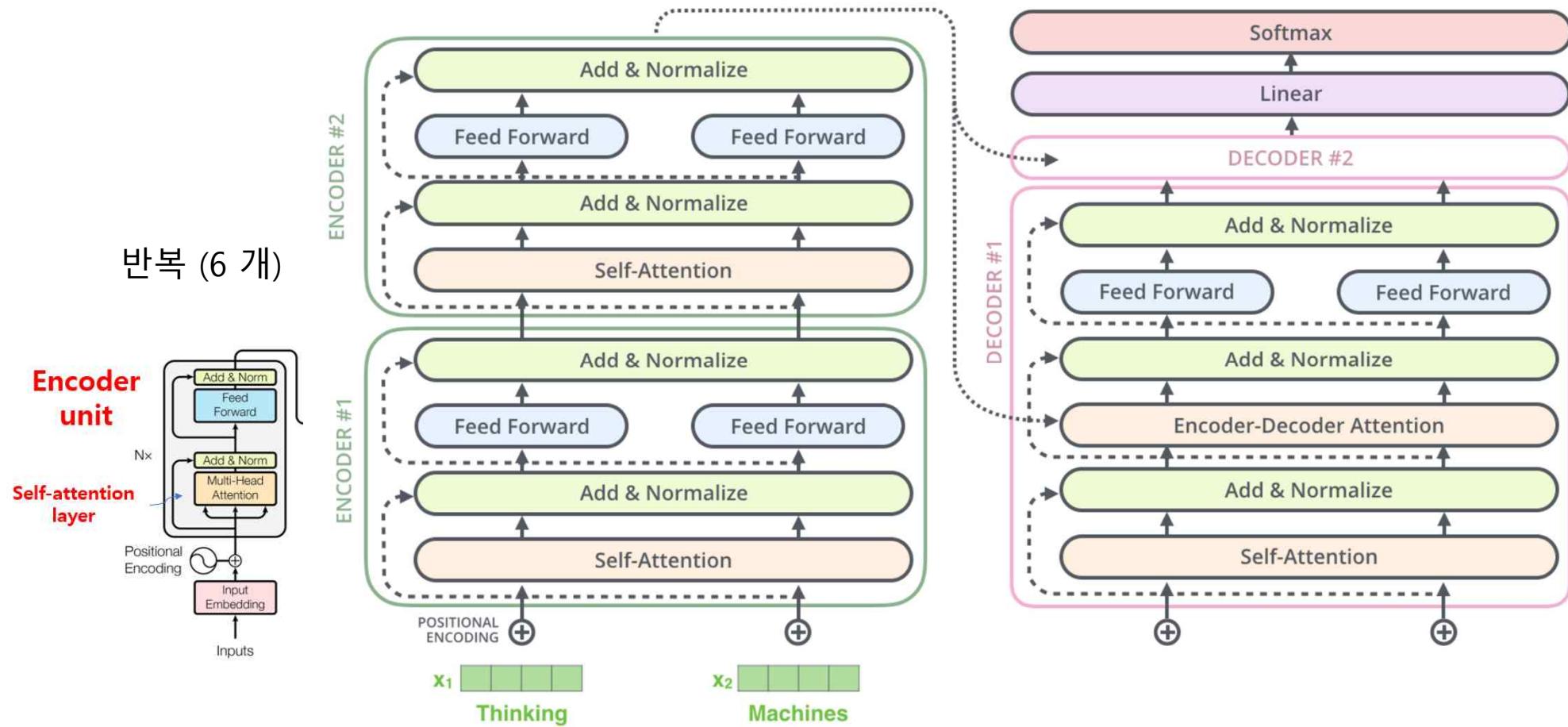
Word Embedding
동시 생성



Transformer Architecture



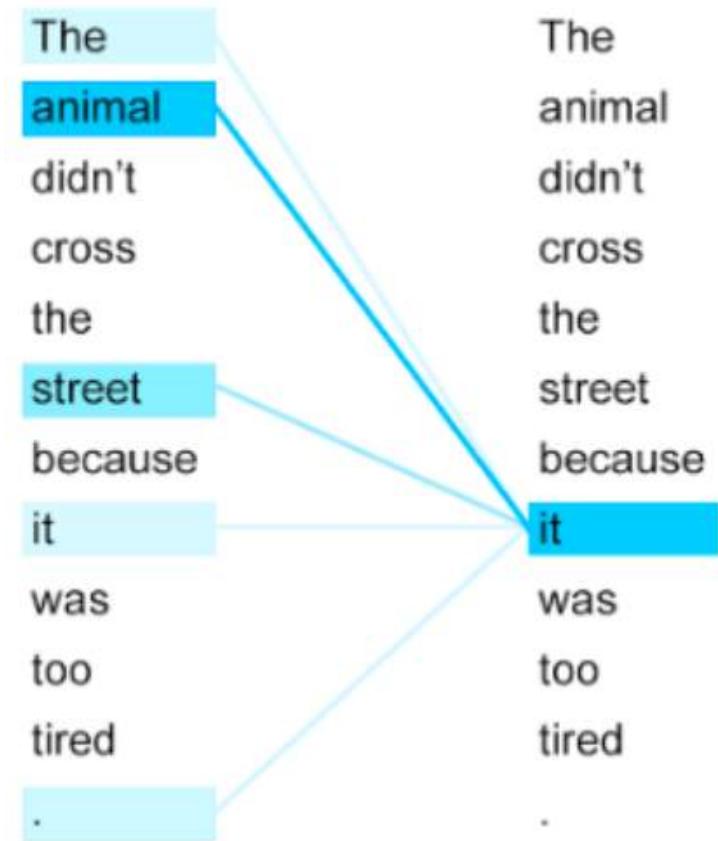
- 기존의 encoder-decoder 와 동일한 구조 유지



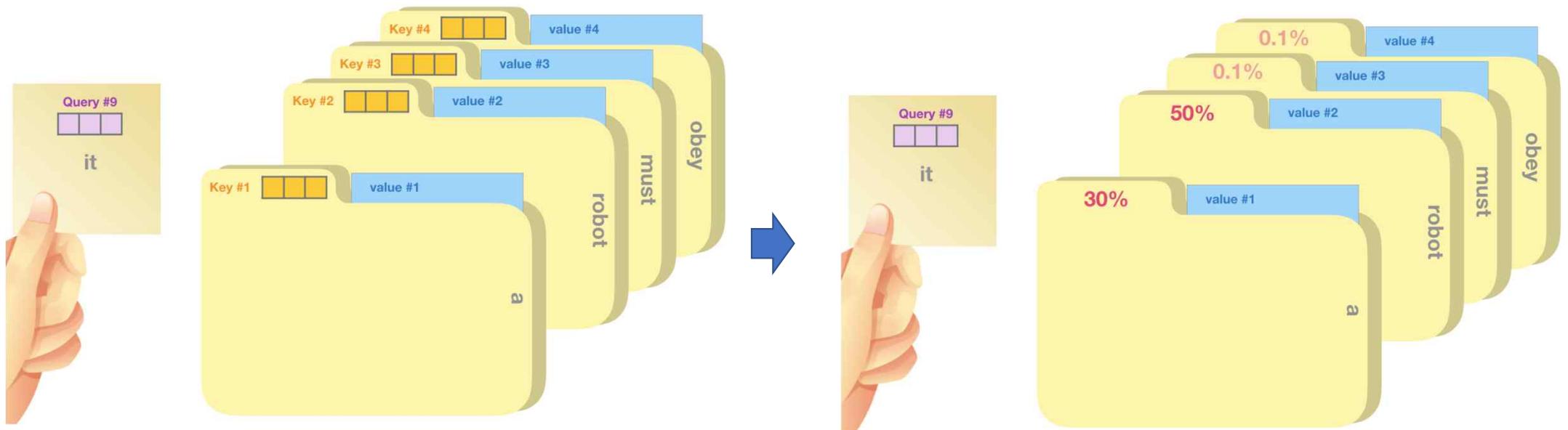
Self-Attention (Intra-Attention)

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

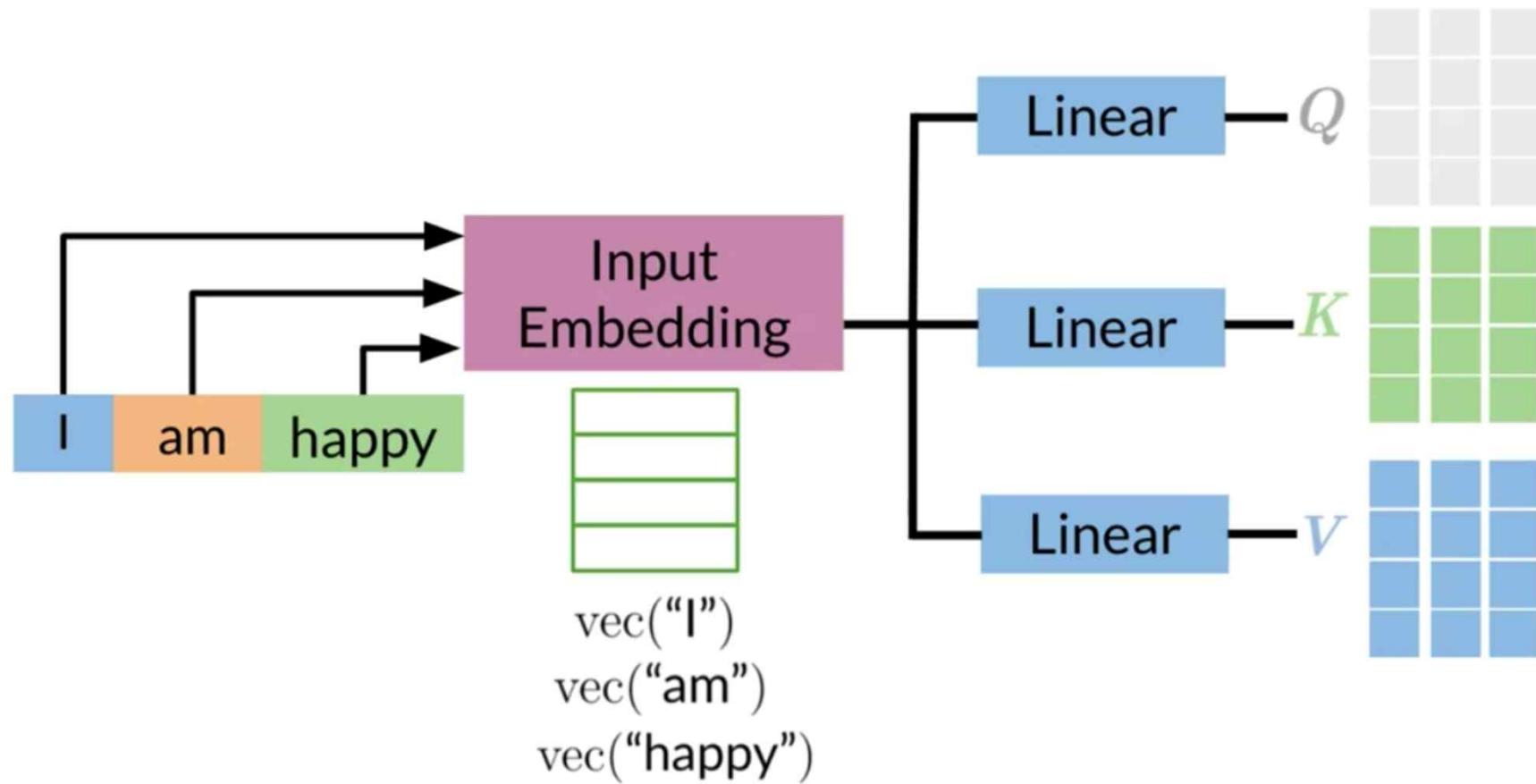


Self-Attention 과 Query, Key, Values

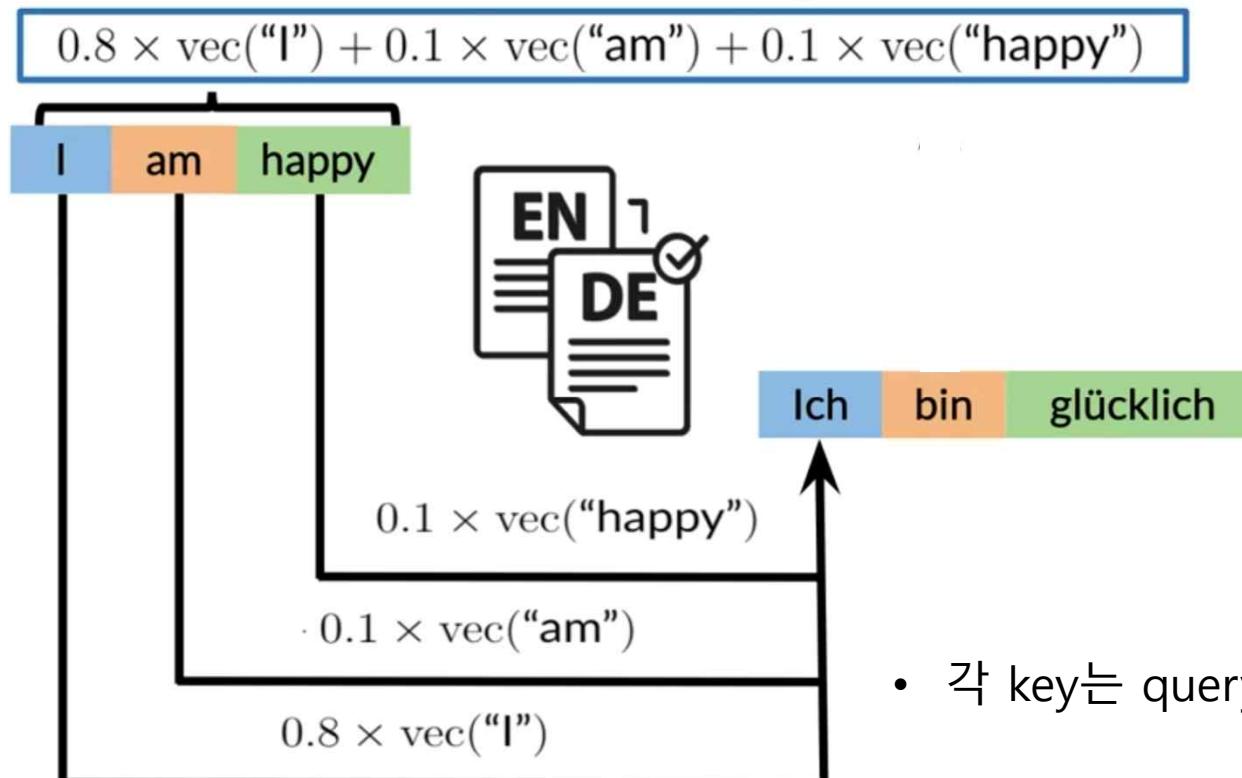


Query, Keys, Values

weight matrix



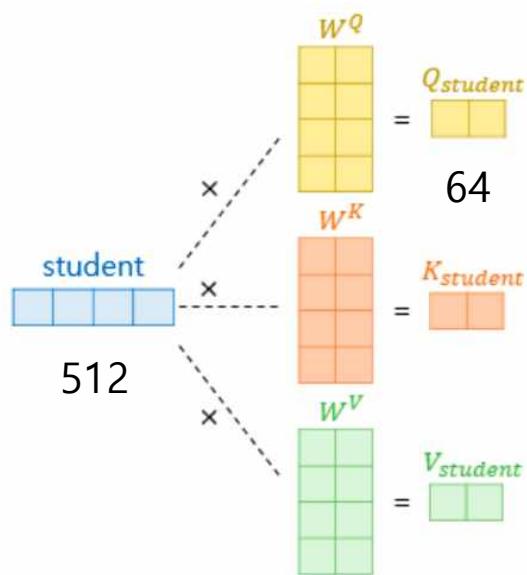
Result of Attention



- 영어 → 독일어 번역의 경우, query (독일어 단어) 는 유사한 key (영어 단어) 를 찾는다.
- 각 key는 query에 match 될 확률을 가지고 있다.

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 ($512 / 8 = 64$)



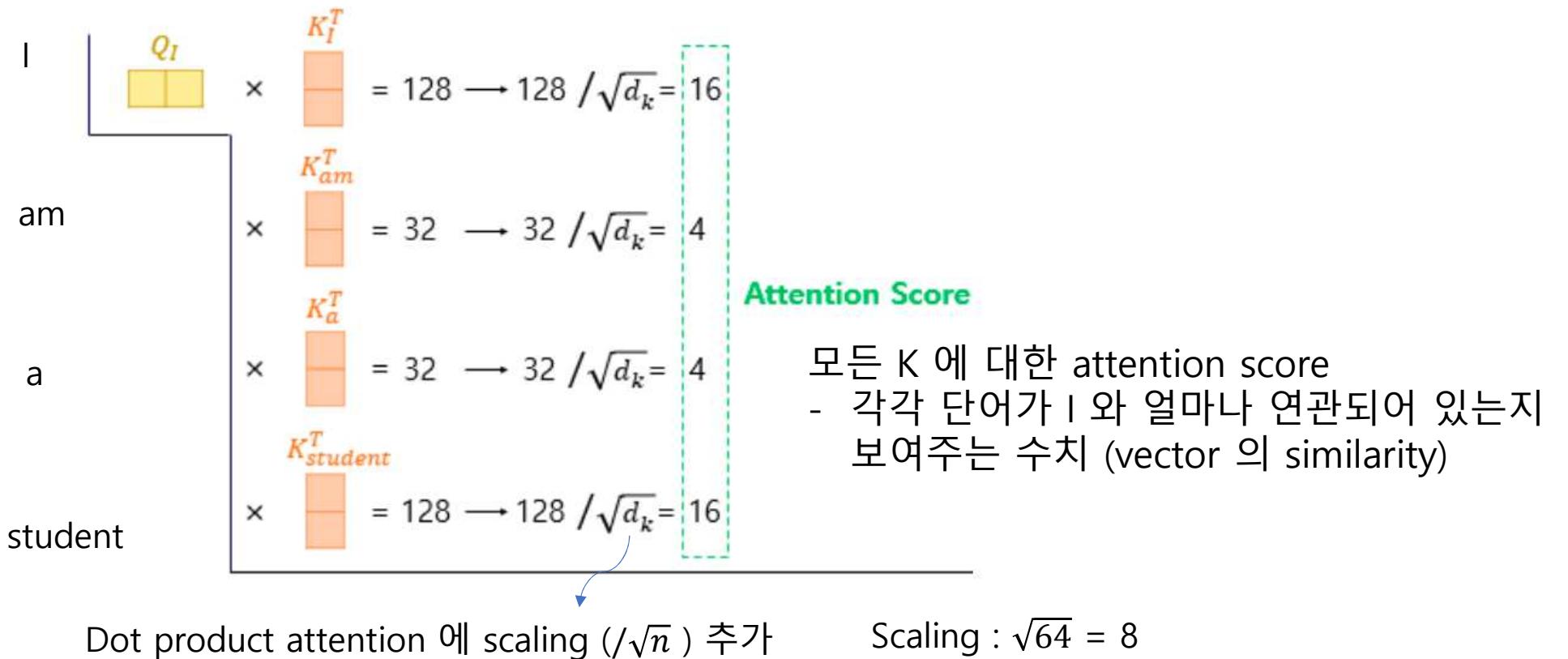
가중치 행렬 W_q , W_k , W_v 는 훈련과정에서 학습

Linear Layer

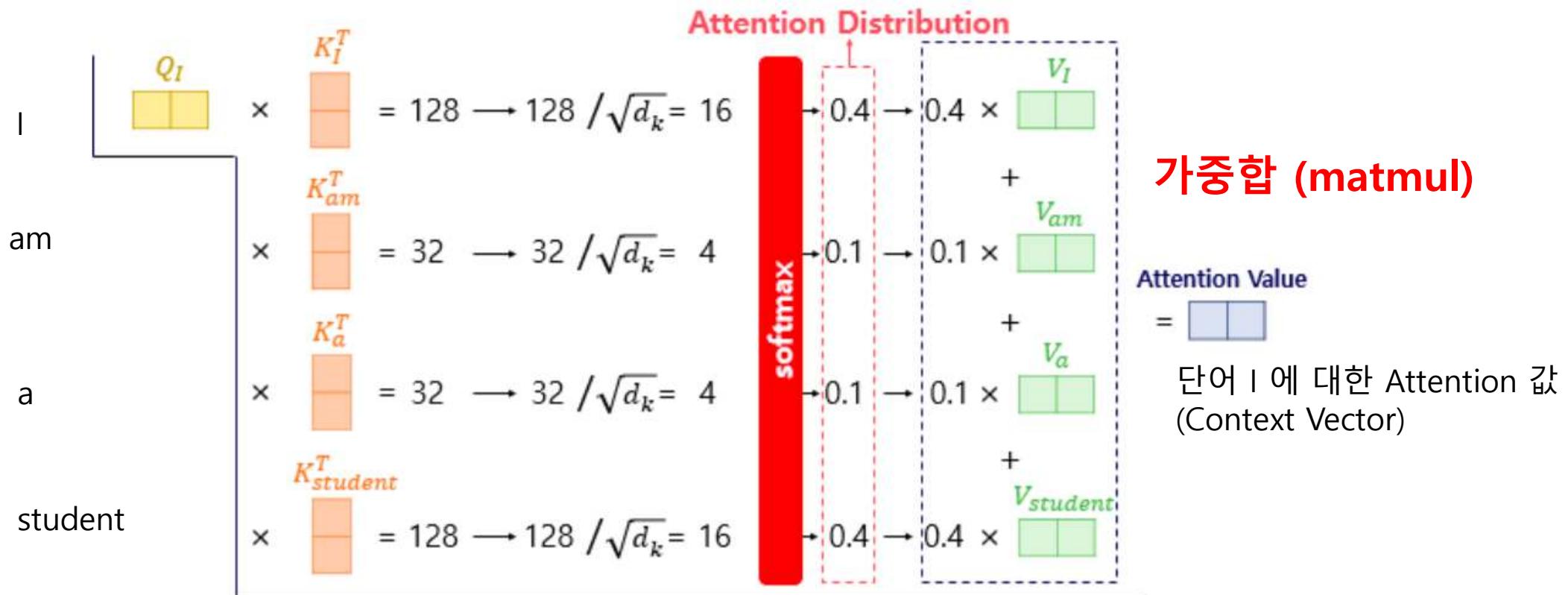
```
self.wq = tf.keras.layers.Dense(d_model)
self.wk = tf.keras.layers.Dense(d_model)
self.wv = tf.keras.layers.Dense(d_model)
```

- 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)

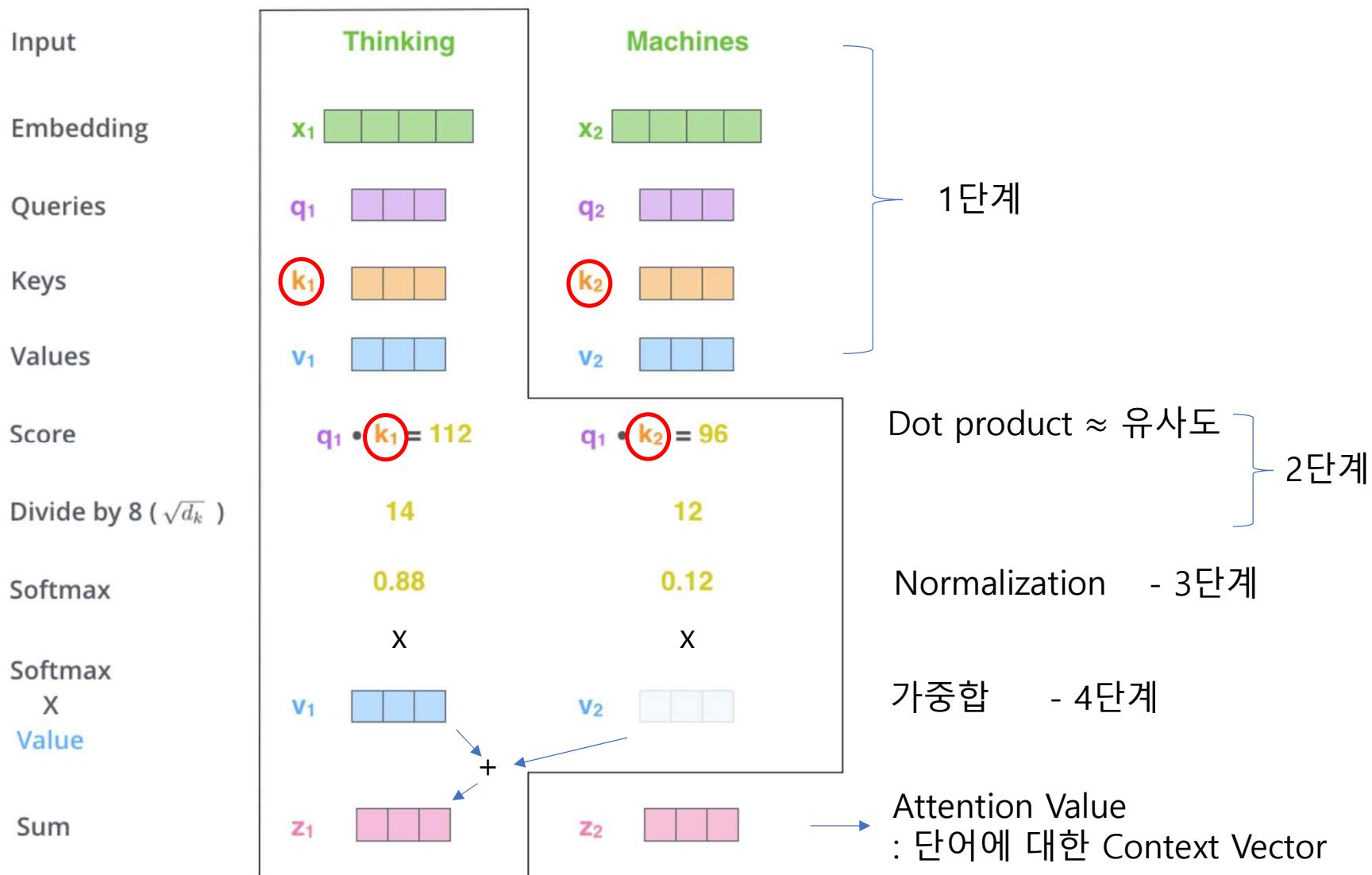


- 3 단계 – Softmax 로 Attention 분포를 구한 후
- 4 단계 - 이를 이용하여 V vector 를 가중합 하여 Attention Value 를 구함



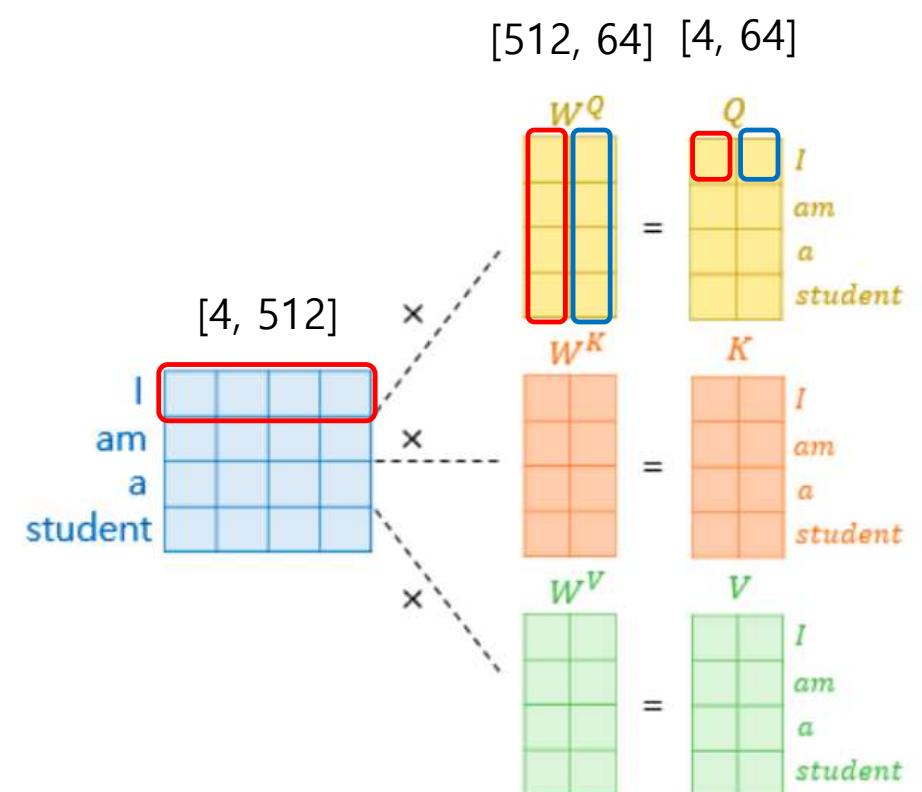
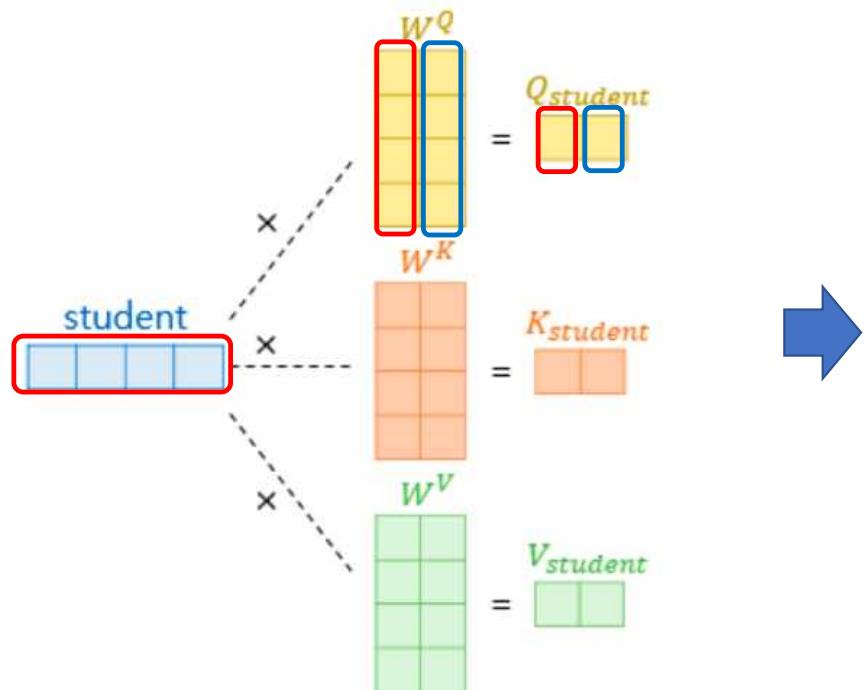
Self-Attention 계산

All-together



행렬 연산으로 일괄 처리

- 위의 과정을 Matrix 연산 처리



Scaled dot product attention formula

$$\begin{matrix} Q \\ l \\ am \\ a \\ student \end{matrix} \times K^T = \begin{matrix} I & am & a & student \\ I & 0.7 & 0.1 & 0.1 & 0.1 \\ am & 0.1 & 0.6 & 0.2 & 0.1 \\ a & 0.1 & 0.3 & 0.6 & 0.1 \\ student & 0.1 & 0.3 & 0.3 & 0.3 \end{matrix}$$

$$\downarrow \quad \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{softmax}\left(\frac{\begin{matrix} Q \\ \times K^T \end{matrix}}{\sqrt{d_k}}\right) \times \begin{matrix} V \\ \begin{matrix} I \\ am \\ a \\ student \end{matrix} \end{matrix} = \begin{matrix} \text{Attention Value} \\ \text{Matrix } a \\ \begin{matrix} I \\ am \\ a \\ student \end{matrix} \end{matrix}$$

각 단어의 attention 값을
모두 가지는 attention 값
행렬 (seq_len, d_v)

seq_len : 입력문장의 길이
 d_v - V vector 의 크기

(참고) scaled_dot_product_attention 구현

```
matmul_qk = tf.matmul(q, k, transpose_b=True)
# scale matmul_qk
dk = tf.cast(tf.shape(k)[-1], tf.float32)
scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

# scaled_attention_logit 에 (mask * -1e9) 를 더하여 padding 위치 (masked 1) 의 값이 0 이 되도록 함
if mask is not None:
    scaled_attention_logits += (mask * -1e9)

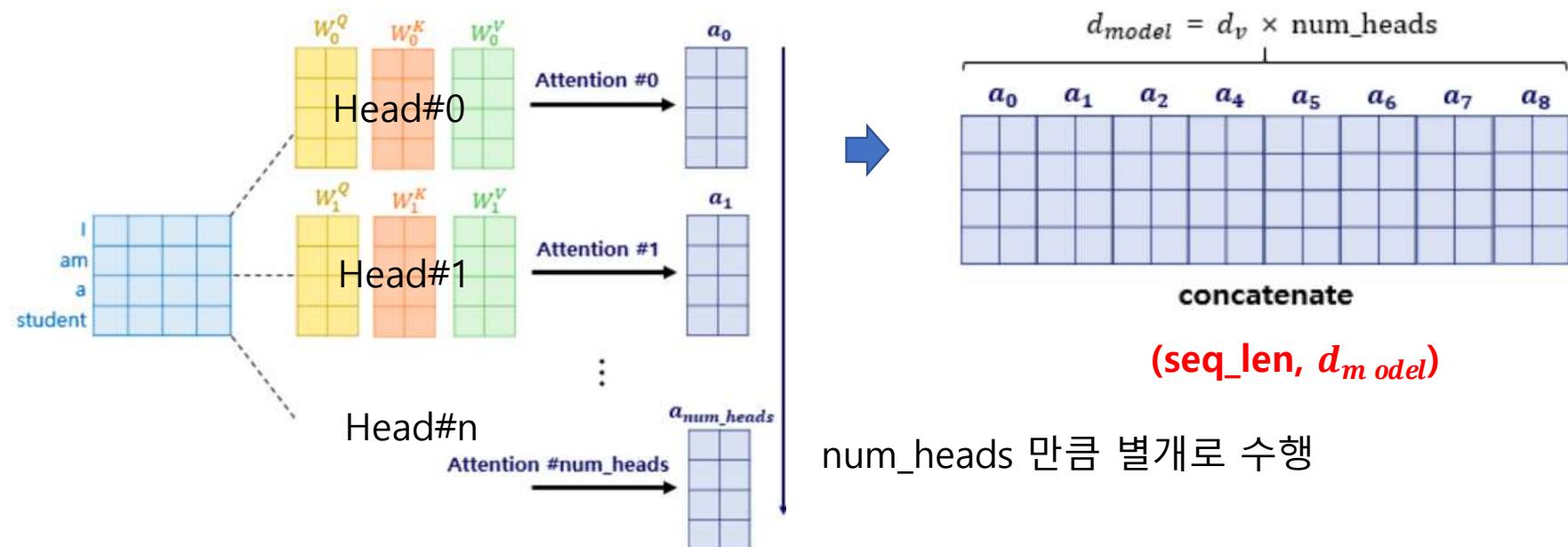
# last axis (seq_len_k) 를 기준으로 softmax normalize 되어 scores 의 합이 1 이 되도록 함.
attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1) # [..., seq_len_q, seq_len_k]

output = tf.matmul(attention_weights, v) # [..., seq_len_q, depth_v]
```

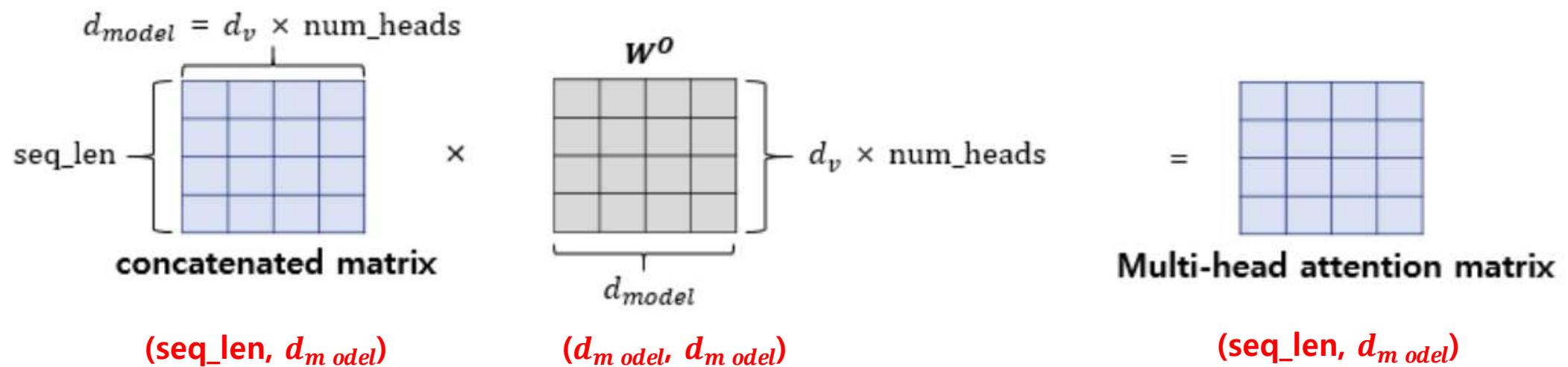
```
q = self.wq(q) # (batch_size, seq_len, d_model)
k = self.wk(k) # (batch_size, seq_len, d_model)
v = self.wv(v) # (batch_size, seq_len, d_model)
```

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

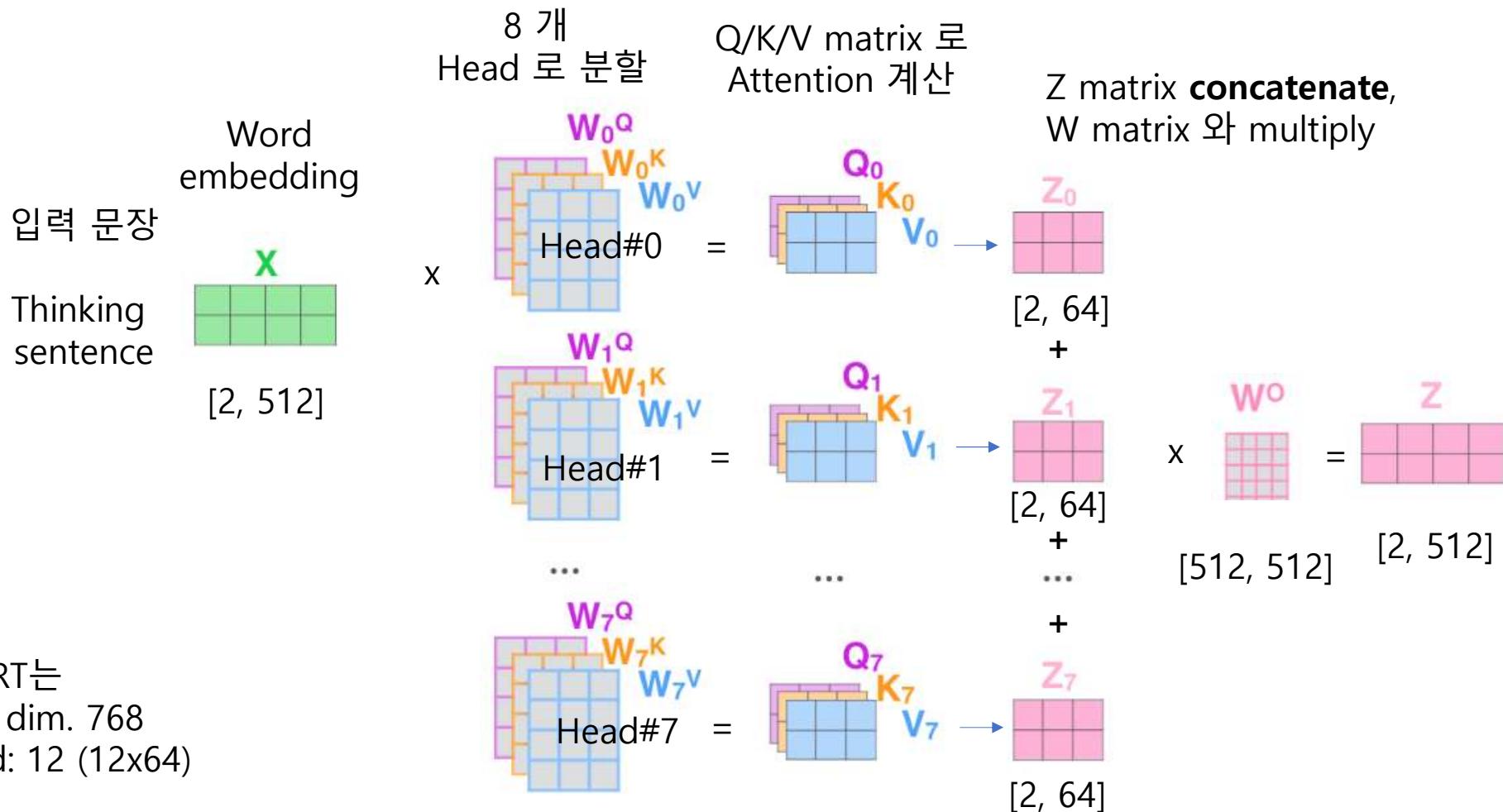


Multi-Head Attention output matrix



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

All together



* BERT는
EMB dim. 768
Head: 12 (12x64)

(참고) MultiHeadAttention 구현

```
batch_size = tf.shape(q)[0]

q = self.wq(q) # (batch_size, seq_len, d_model)
k = self.wk(k) # (batch_size, seq_len, d_model)
v = self.wv(v) # (batch_size, seq_len, d_model)

q = self.split_heads(q, batch_size) # (batch_size, num_heads, seq_len_q, depth)
k = self.split_heads(k, batch_size) # (batch_size, num_heads, seq_len_k, depth)
v = self.split_heads(v, batch_size) # (batch_size, num_heads, seq_len_v, depth)

# scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
# attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)
scaled_attention, attention_weights = scaled_dot_product_attention(q, k, v, mask)

scaled_attention = tf.transpose(scaled_attention,
                               perm=[0, 2, 1, 3]) # (batch_size, seq_len_q, num_heads, depth)

concat_attention = tf.reshape(scaled_attention,
                             (batch_size, -1, self.d_model)) # (batch_size, seq_len_q, d_model)

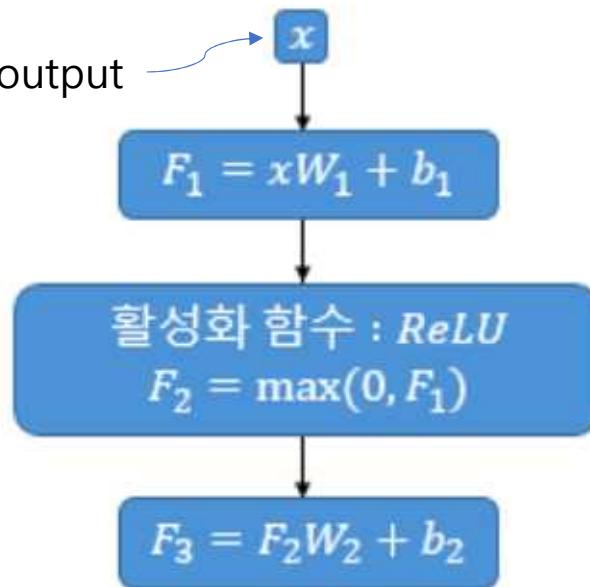
output = self.dense(concat_attention) # (batch_size, seq_len_q, d_model)
```

Position-wise Feed Forward NN

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

$$FFNN(x) = \text{MAX}(0, xW_1 + b_1)W_2 + b_2$$

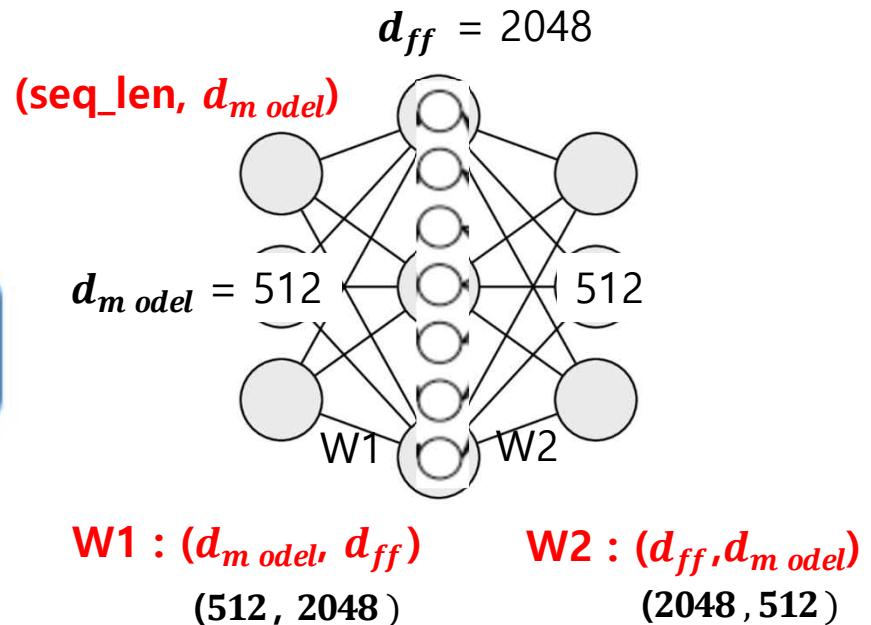
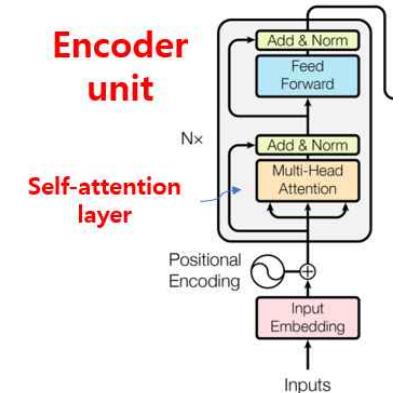
Multi-Head Attention 의 output



참고) BERT

$d_{model} = 768$

$d_{ff} = 3072$ (768×4)

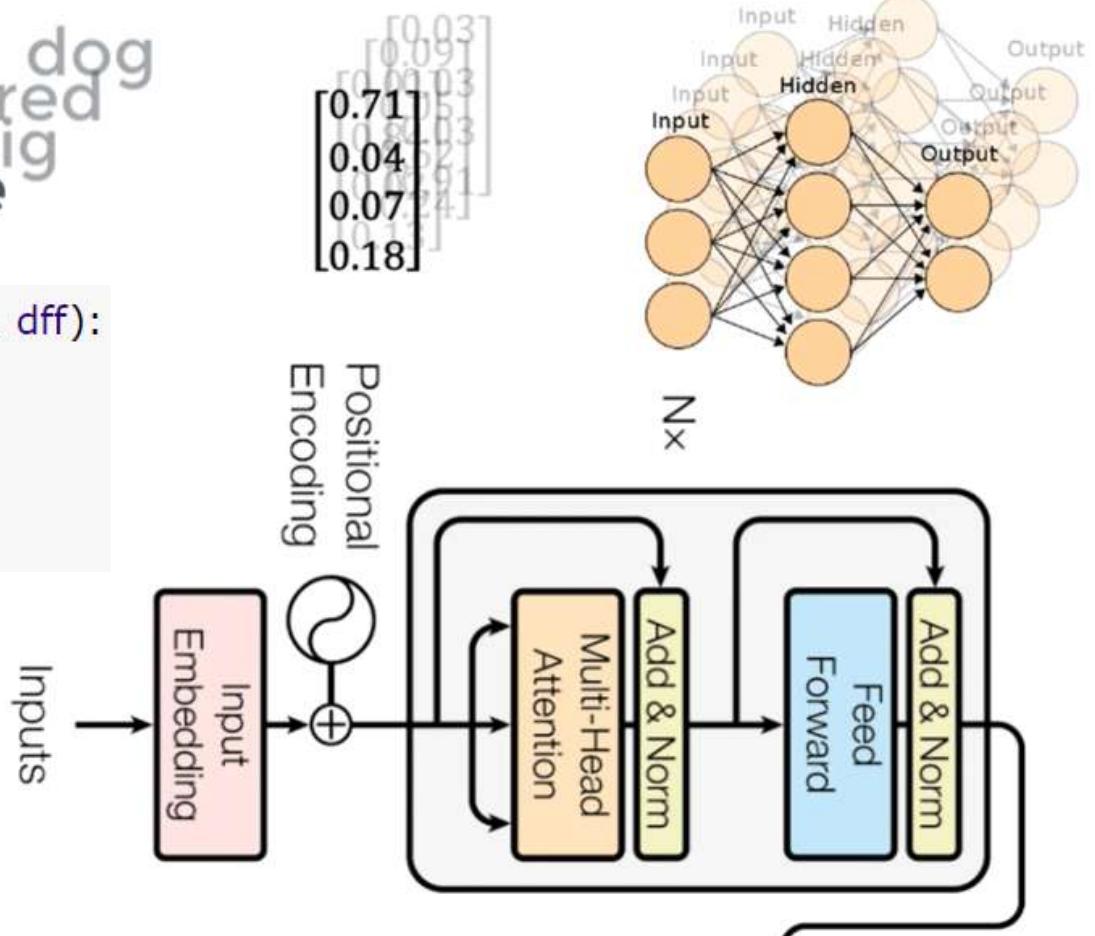


(참고) Position-wise Feed Forward NN 구현

(batch_size, seq_len, dff)

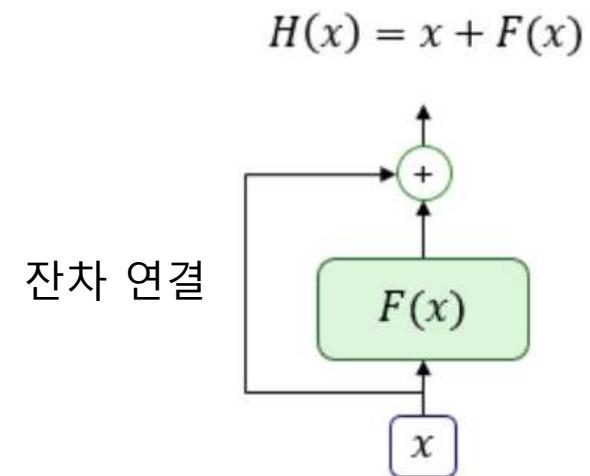
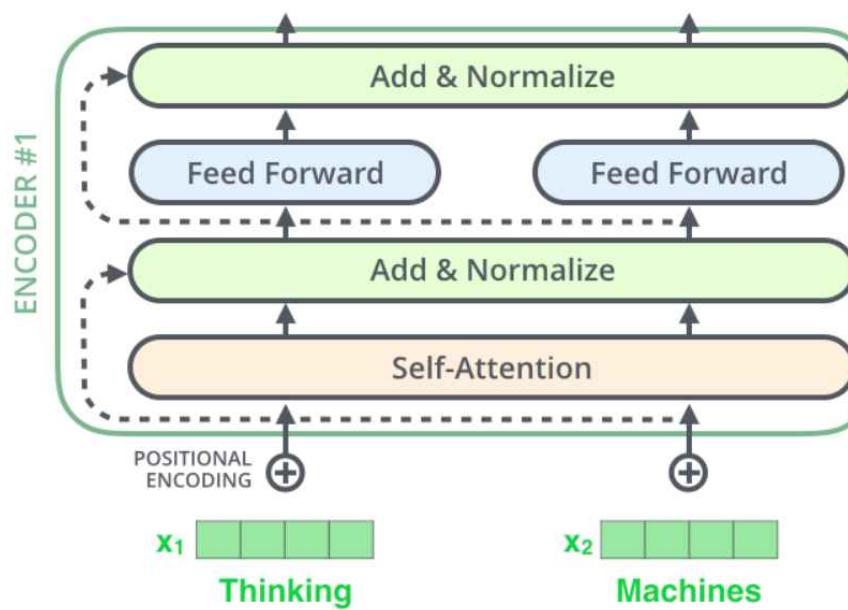
```
def point_wise_feed_forward_network(d_model, dff):
    return tf.keras.Sequential([
        tf.keras.layers.Dense(dff, activation='relu'),
        tf.keras.layers.Dense(d_model)
    ])
```

(batch_size, seq_len, d_model)



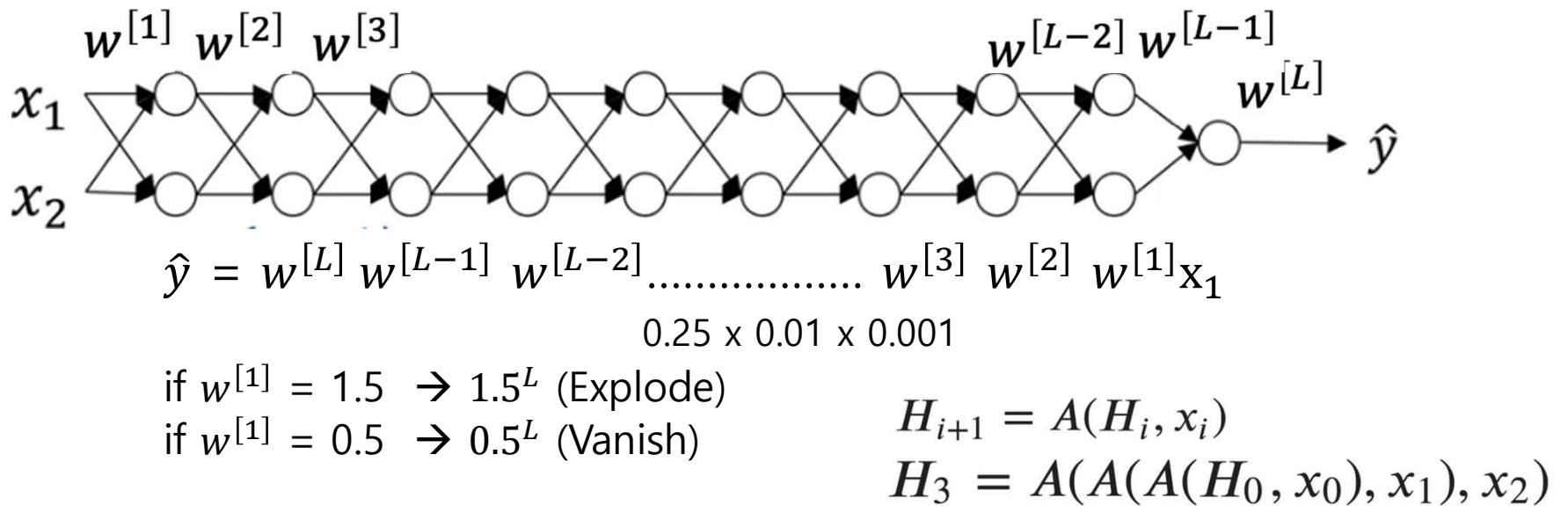
잔차 연결 및 정규화 (Residual Connection and Layer Normalization)

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



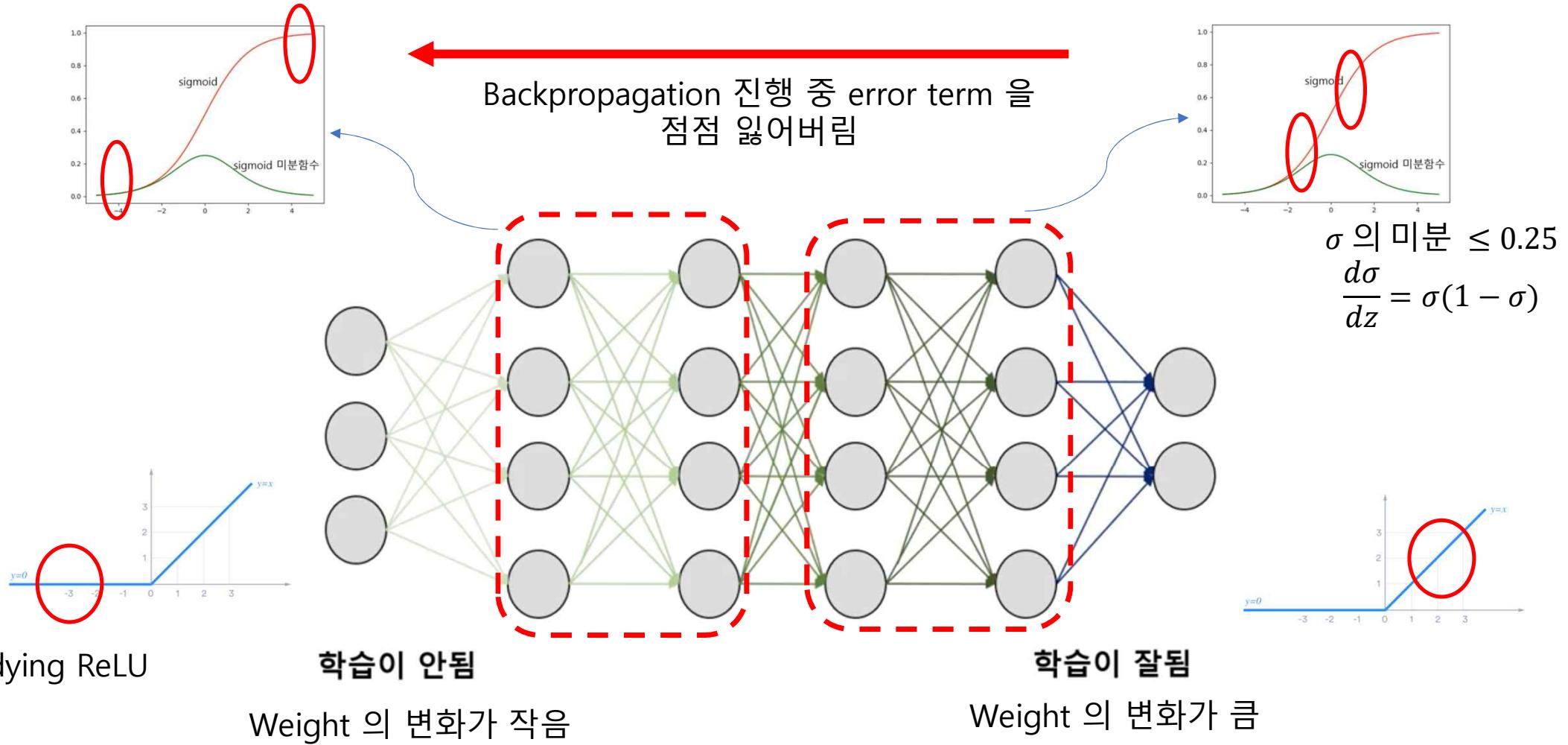
(참고) Vanishing & Exploding Gradient

- 매우 deep 한 network 을 훈련시킬 경우 앞부분 Layer weight 의 미분값 크기가 매우 작아지거나 커지는 현상



- 문제점 – Exploding – 훈련 자체가 안됨
Vanishing – 경사하강법이 매우 느리게 진행

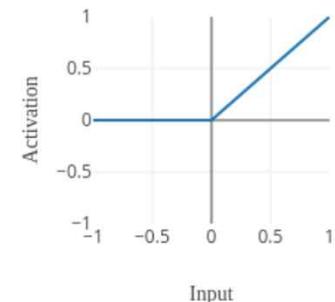
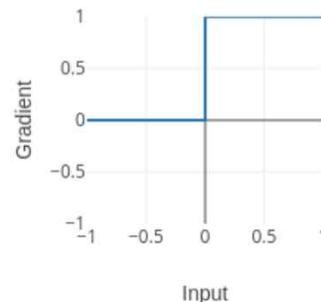
기울기 소실 (Vanishing Gradient)



Solutions of Vanishing Gradient

- Relu 사용

$$\max(0, z) \begin{cases} 1 & f \ z > 0 \\ 0 & f \ z < 0 \end{cases}$$



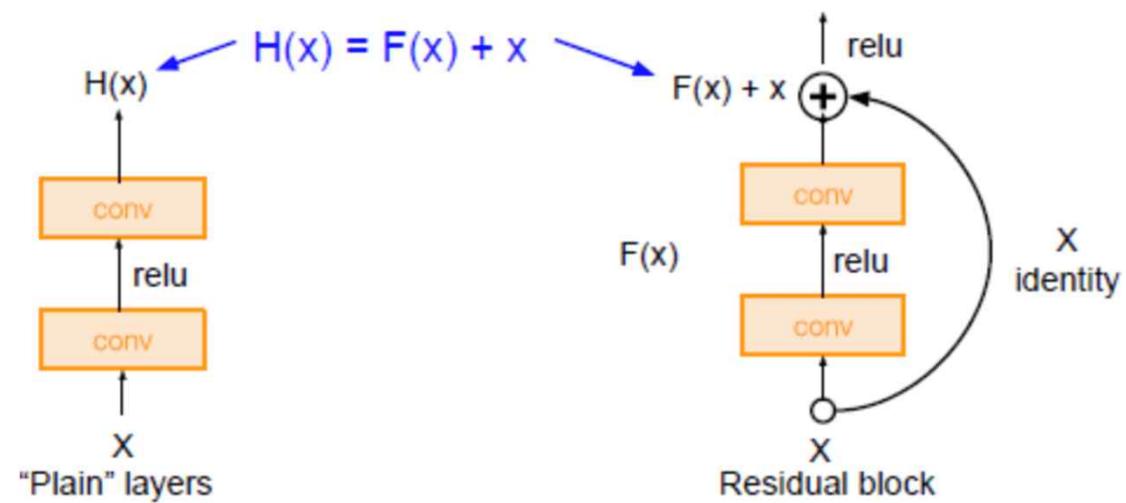
- Weight 의 신중한 초기화

Ex) Xavier (Glorot) Initializer with Tanh, He Initializer with Relu

- Batch Normalization
- Residual Network

ResNet – Identity Shortcut 적용

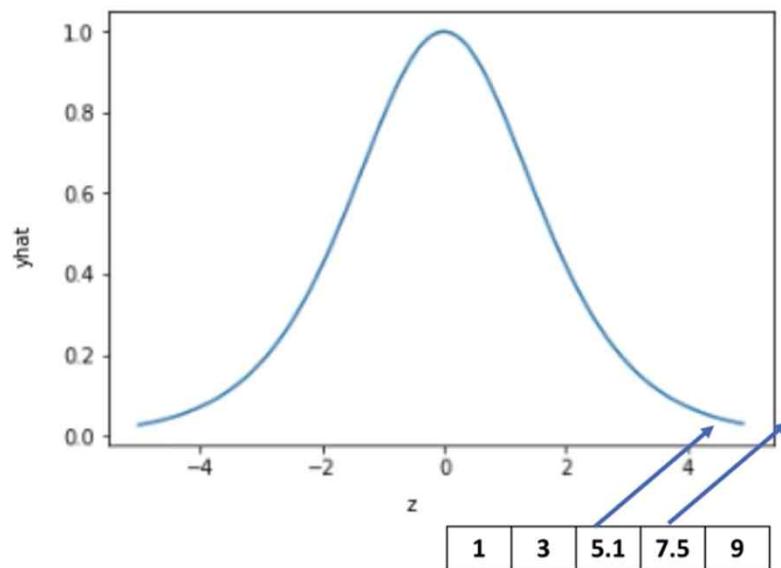
- 2015년 ILSVRC 우승 모델
- 152개 Layer로 구성
- 이전에 학습한 정보를 보존하고, 추가적으로 학습



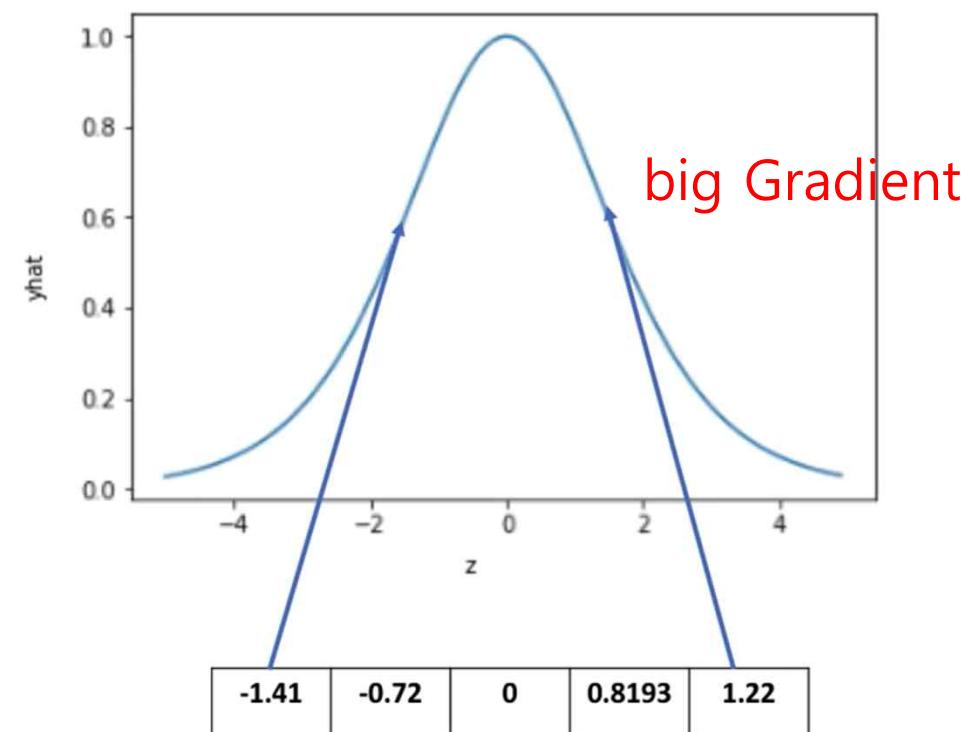
Batch Normalization

- Internal Covariance Shift (내부 공변량 변화)
 - Network의 각 층이나 Activation 마다 input의 distribution이 달라지는 현상
 - 최초 Input Layer 의 normalization 효과가 Hidden Layer 를 거치면서 희석됨
- 각 층의 input의 distribution을 평균 0, 표준편차 1인 input 으로 normalize
- Faster Train

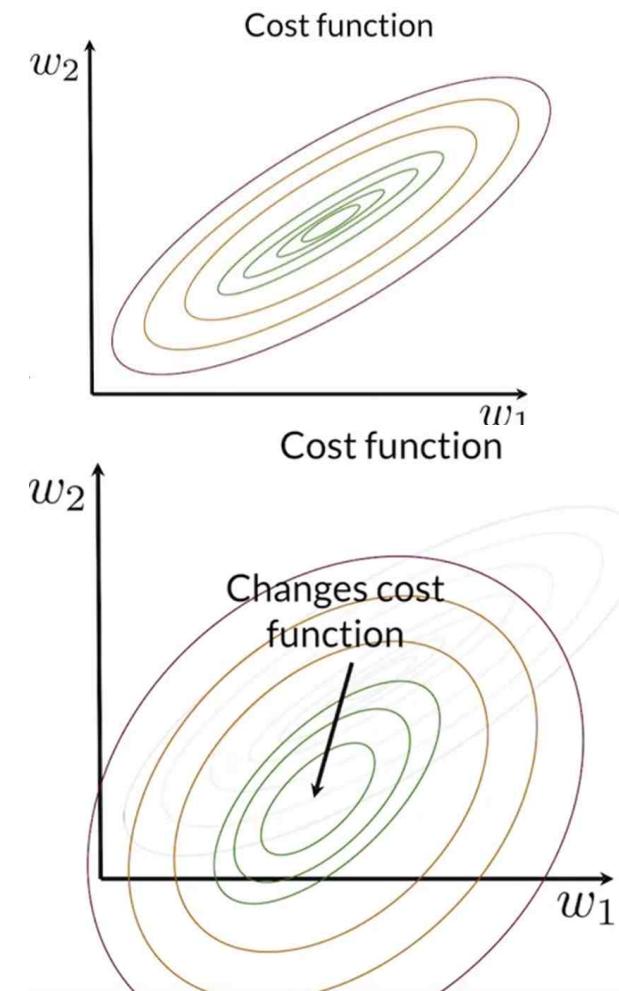
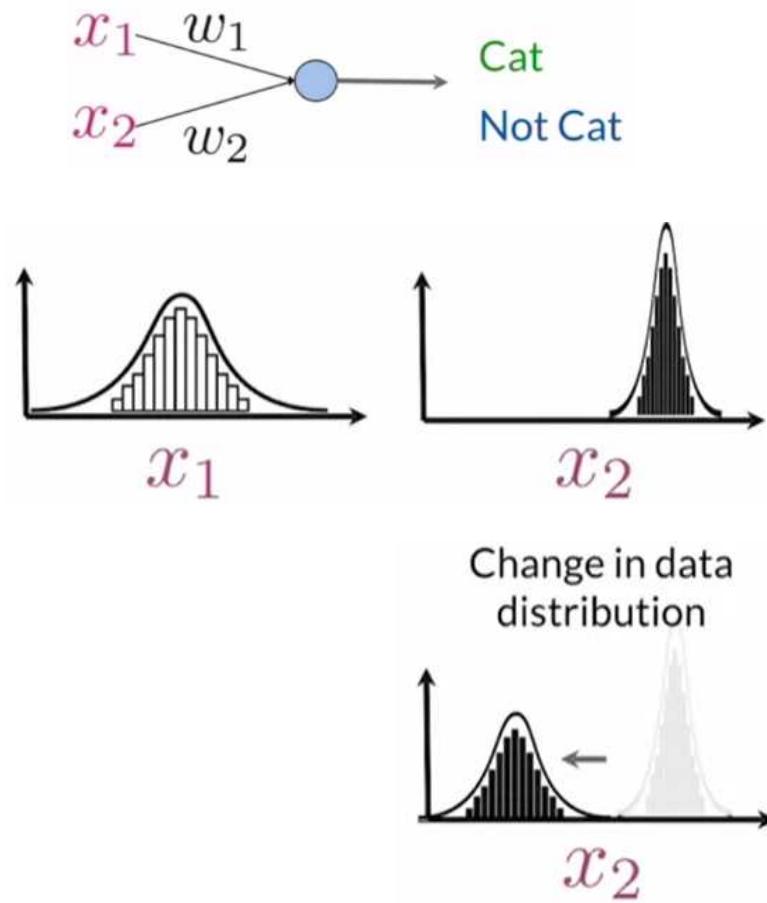
Batch Normalize 전, 후 Gradient 비교



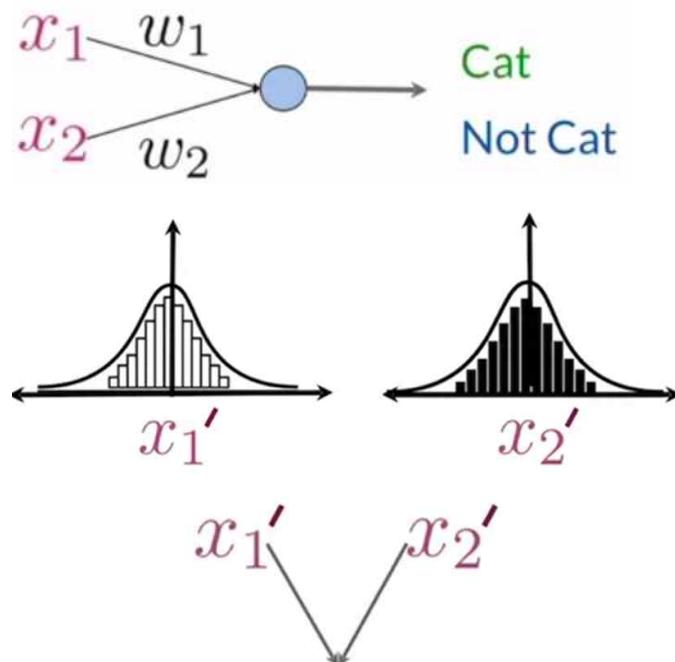
Gradient 소실



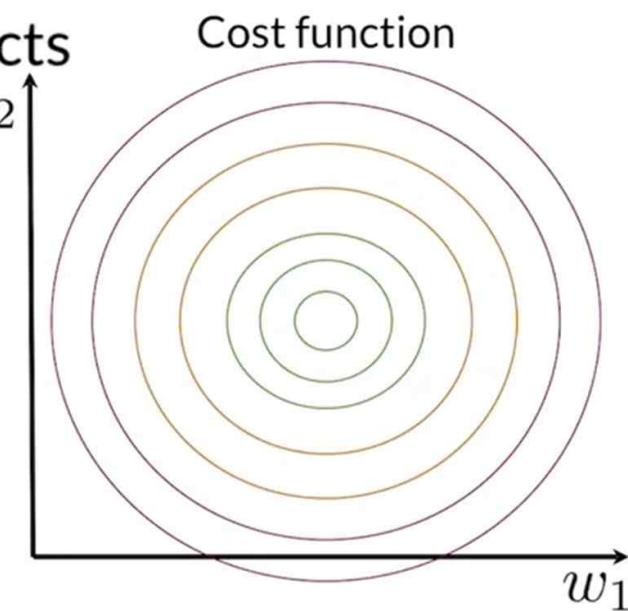
Internal Covariance 와 Batch Normalization



Normalization and Its Effects



Around mean at 0
and std. at 1



Reduction of
Covariance shift

Training data uses
batch stats

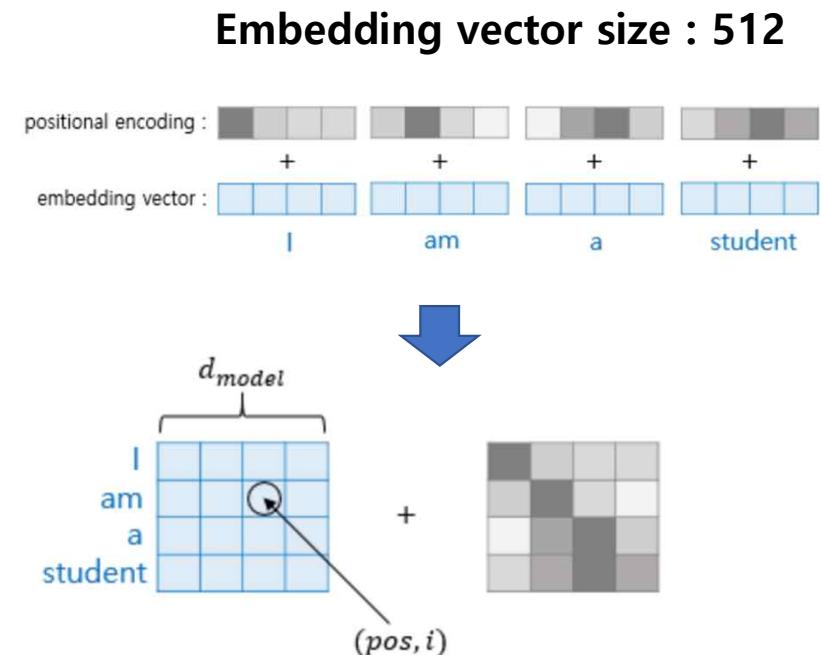
Test data uses
training stats

Transformer 의 주요 Hyper-Parameter

- $d_{model} = 512$
 - 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

Positional Encoding

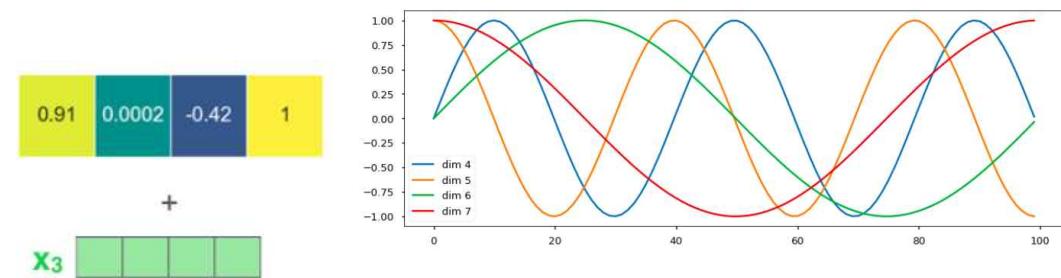
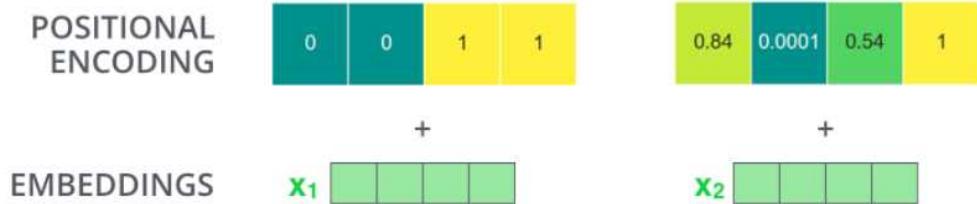
- 위치 정보 특성 해결
- Embedding vector 에 positional encoding 값 추가
- $PE_{(pos,2i)} = \sin(pos / 10000^{2i/d_{model}})$ → 짝수 위치
- $PE_{(pos,2i+1)} = \cos(pos / 10000^{2i/d_{model}})$ → 홀수 위치
 - pos – 입력문장 내의 embedding vector 의 위치
 - i – embedding vector 내의 차원의 index



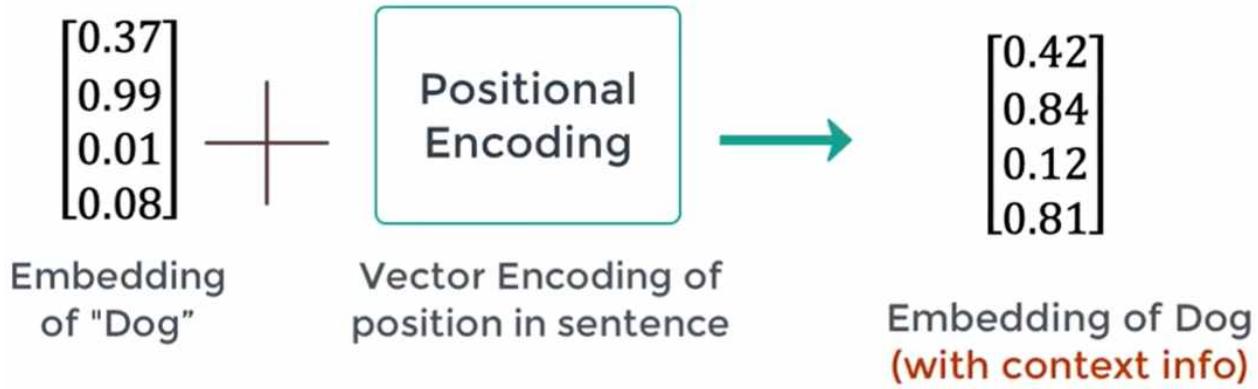
- 같은 단어라도 문장 내의 위치에 따라서 Transformer의 입력으로 들어가는 Embedding vector 값이 달라진다 → 순서의 정보가 포함된 Embedding Vector
- 정현파는 일정한 cycle로 값이 규칙적으로 반복되므로, 신경망이 상대적 위치 값을 쉽게 학습

[유튜브 설명 동영상](#)

Positional Encoding example

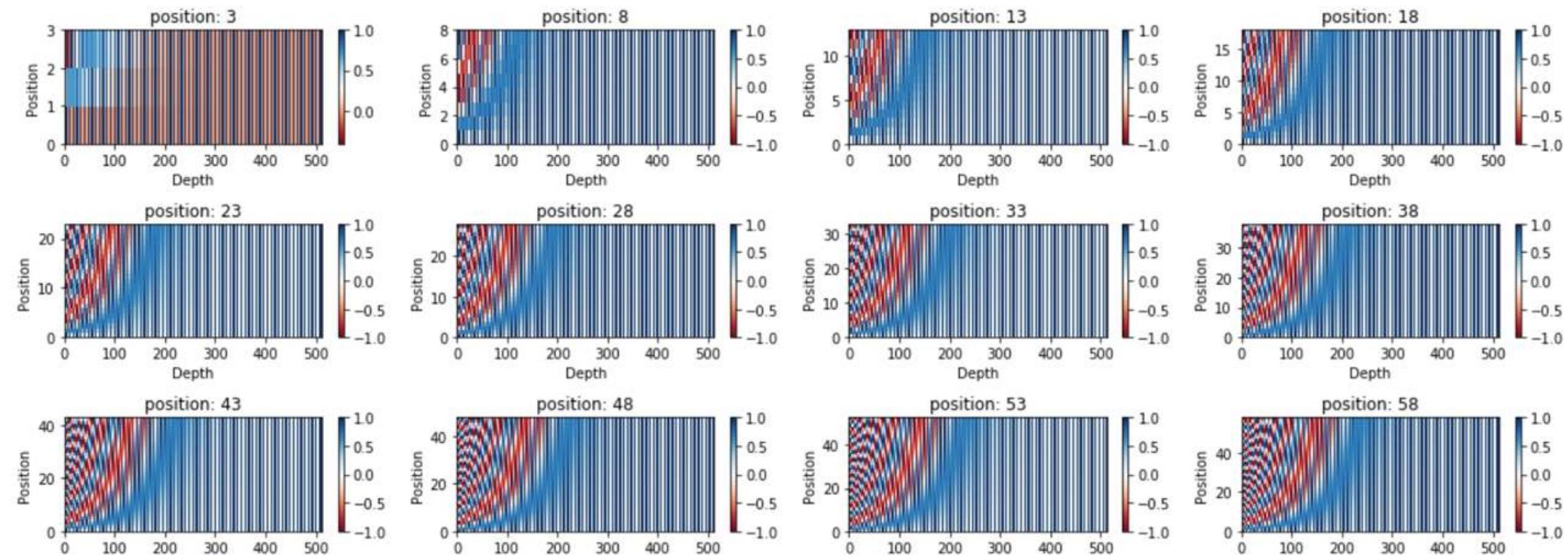


INPUT Je suis étudiant



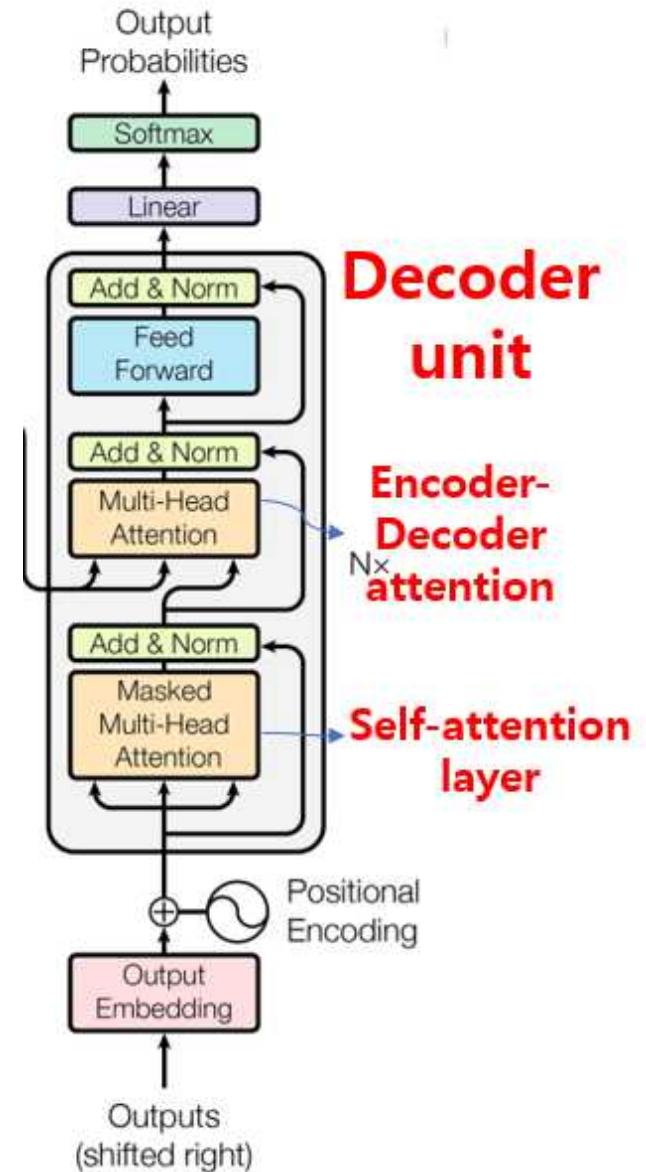
참고) BERT
Learnable Positional
Embedding 사용

Positional Encoding Value 시각화

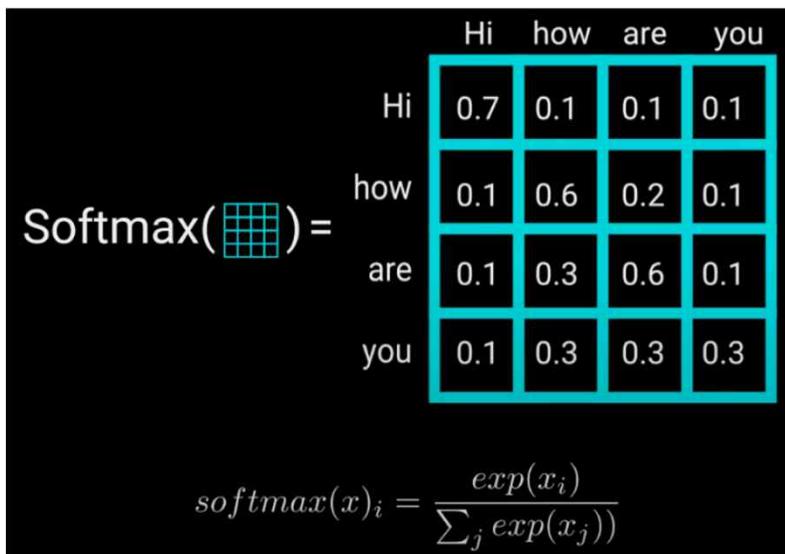


Decoder

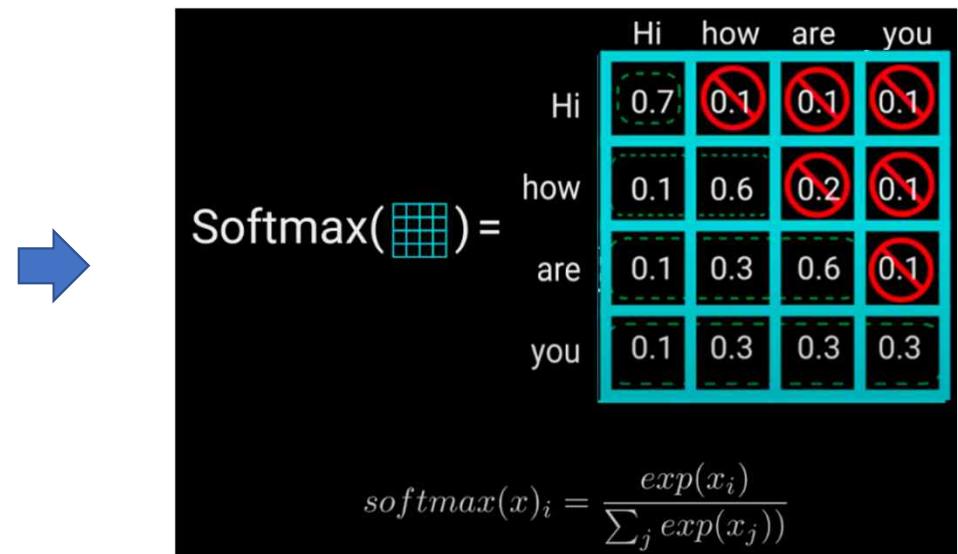
- Top Encoder 의 output 이 Decoder 의 attention vector **Key, Value** 로 transform 되어 각 decoder 의 “encoder-decoder attention” layer 에서 사용
→ Input sequence 의 적절한 위치를 focus 하도록 함
- **Query** : training - ground truth label 사용
inference – previous predictions 사용
- Decoding result 는 encoder 경우처럼 위로 전달
→ 각 step 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.

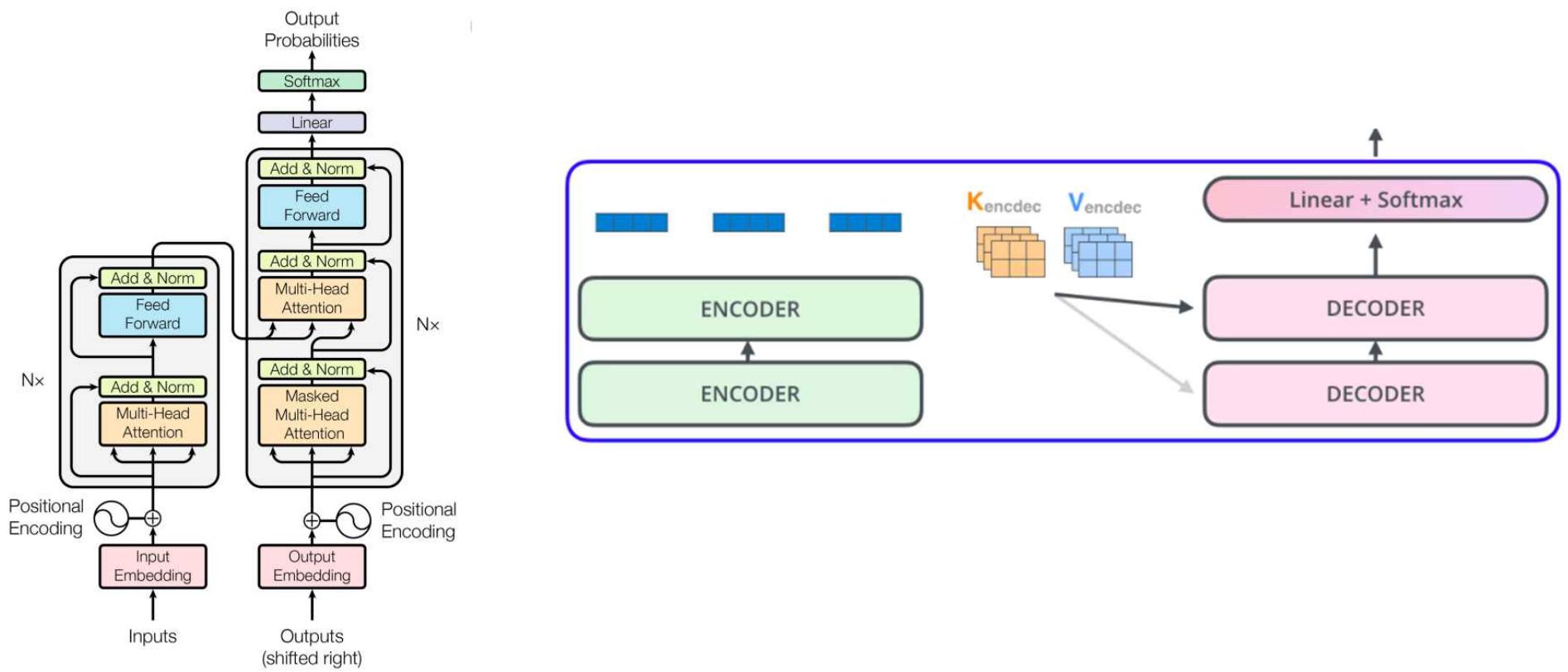


Encoder 의 attention score 계산

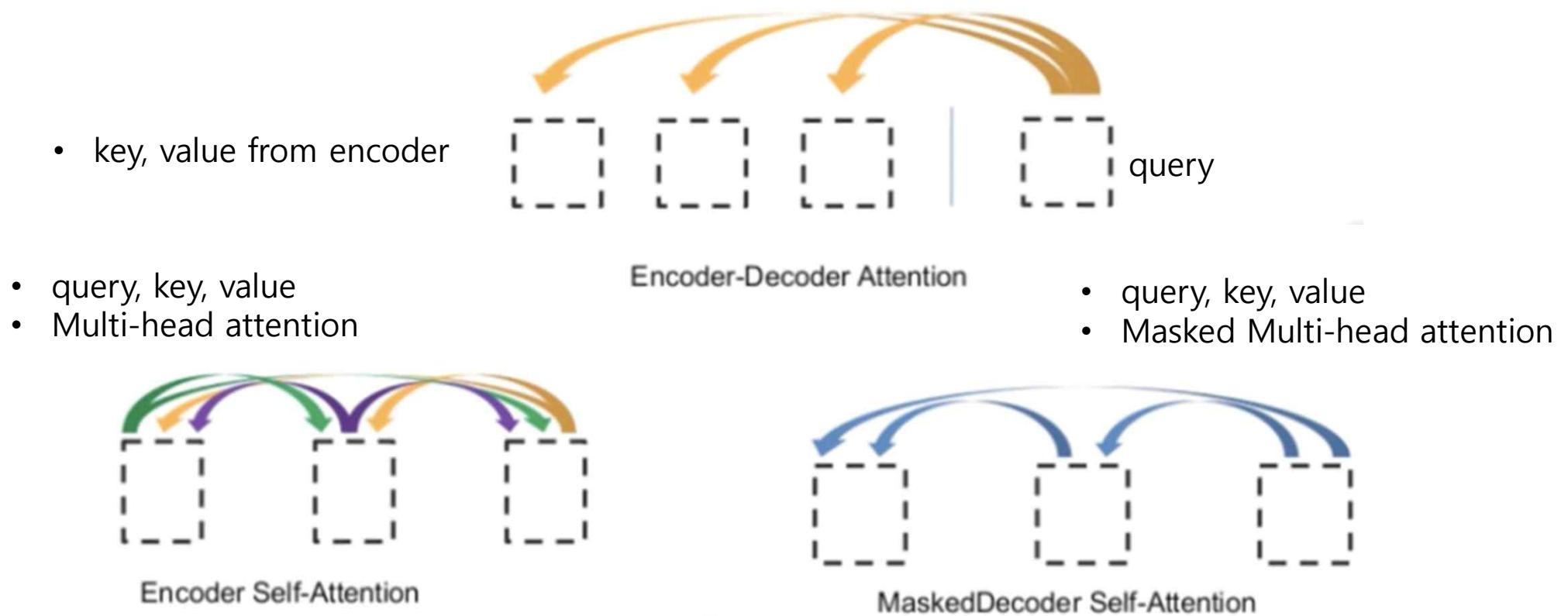


Decoder 의 attention score 계산

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



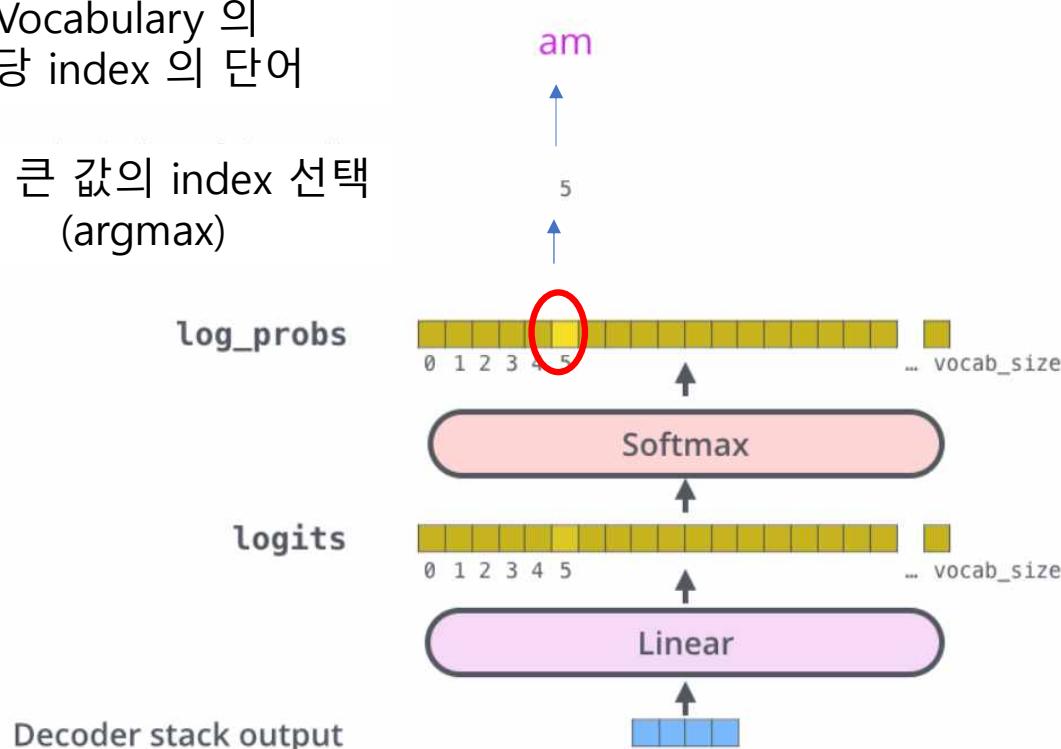
Transformer 의 3 가지 Attention 정리



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

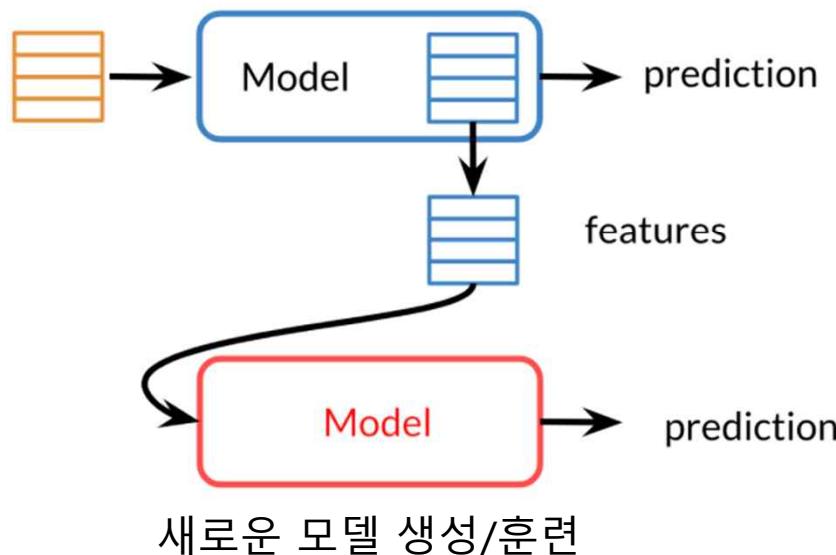
- Training Time 절감
 - GPT-3 와 같은 최첨단 model 은 train 에 약 2 개월 소요
- 예측 성능 향상
- Small dataset 으로 첨단 performance 구현

전이 학습 (Transfer Learning) 방법

Feature-based Transfer Learning

pre-trained model의 출력 feature를
new model 의 입력으로 사용

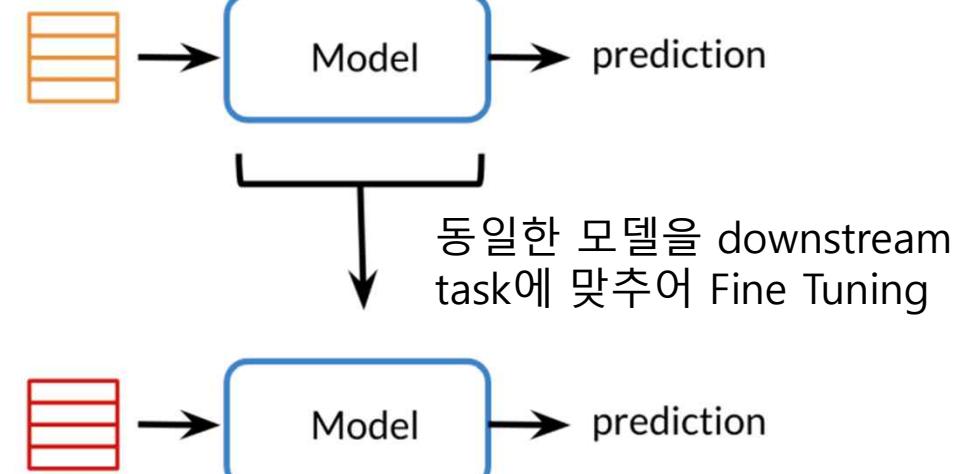
Pre-Train



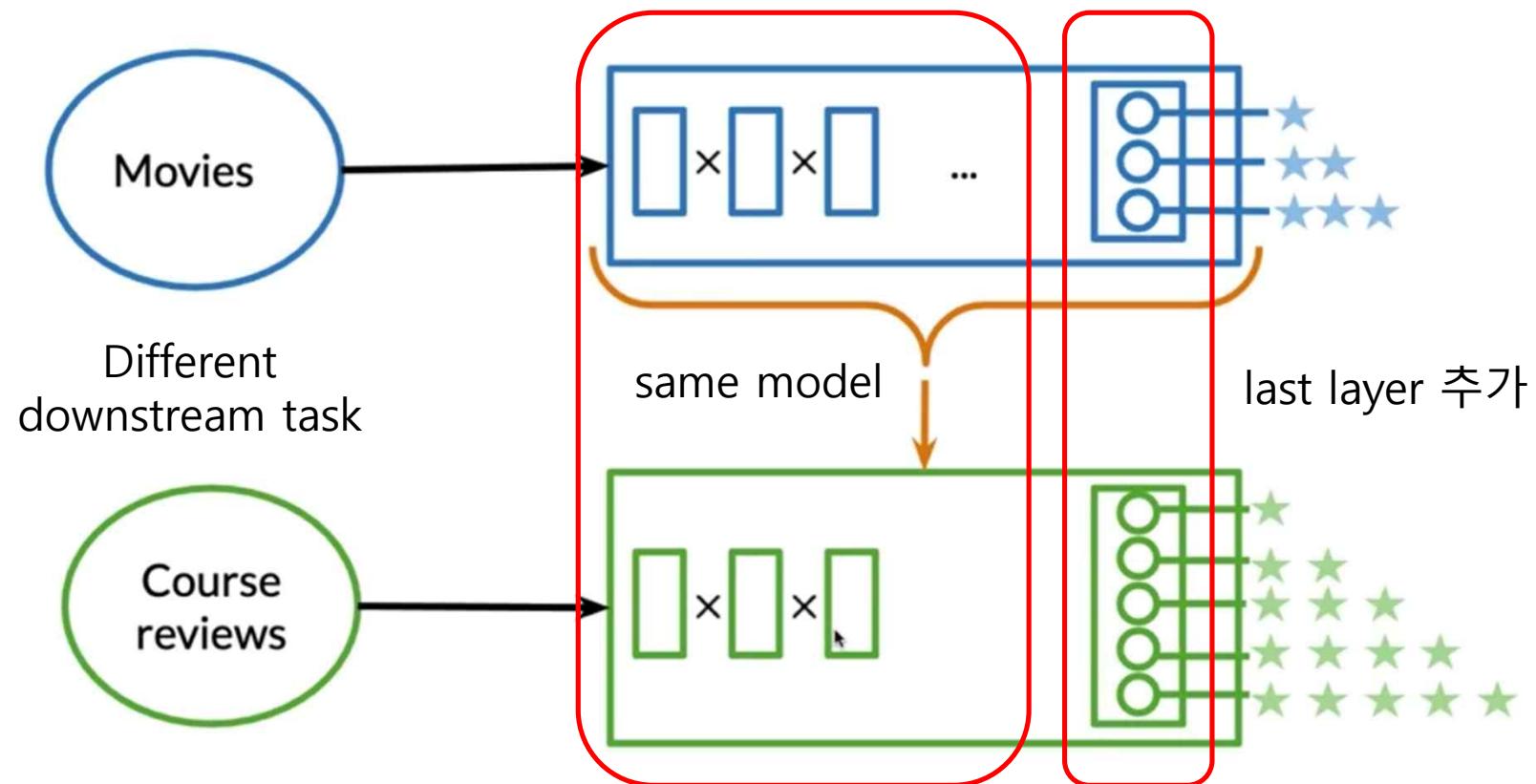
Fine-Tuning

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

Pre-Train



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 domain에 특화된 단어를 모름



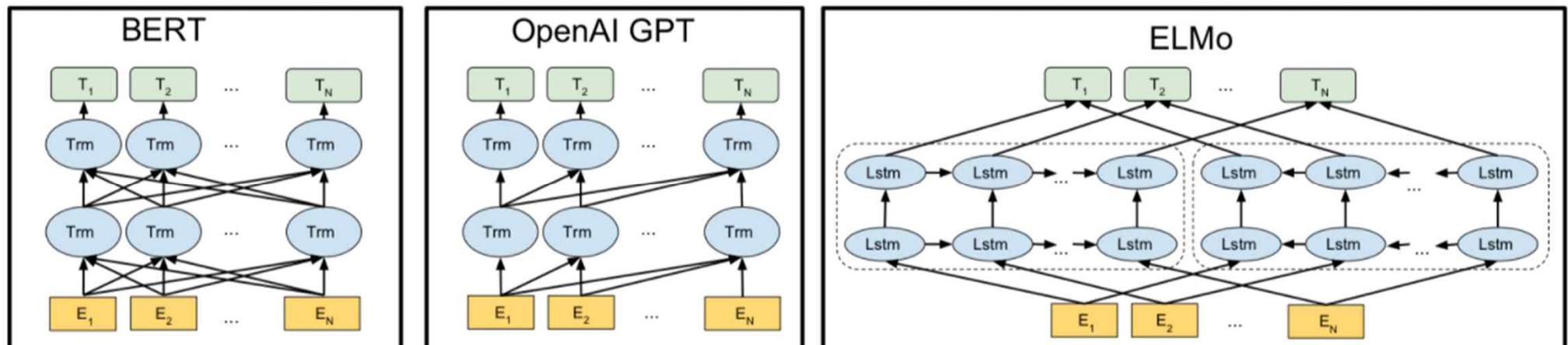
Layer	Weights
Embedding Layer	~24M
Transformers (x12)	~7M ea. x 12 = ~85M
TOTAL	~109M

(417 MB)

Pre-trained Models 비교

Language Model

n 개의 단어를 가지고 뒤의 단어를 예측 (n-gram 방식)



Masked Language Model(MLM) : 주변 단어의 context만을 보고 mask된 단어를 예측



Next Sentence Prediction : 두 문장이 이어지는 문장인지 아닌지 맞추는 것

BERT

(Bidirectional Encoder Representations
from Transformers)

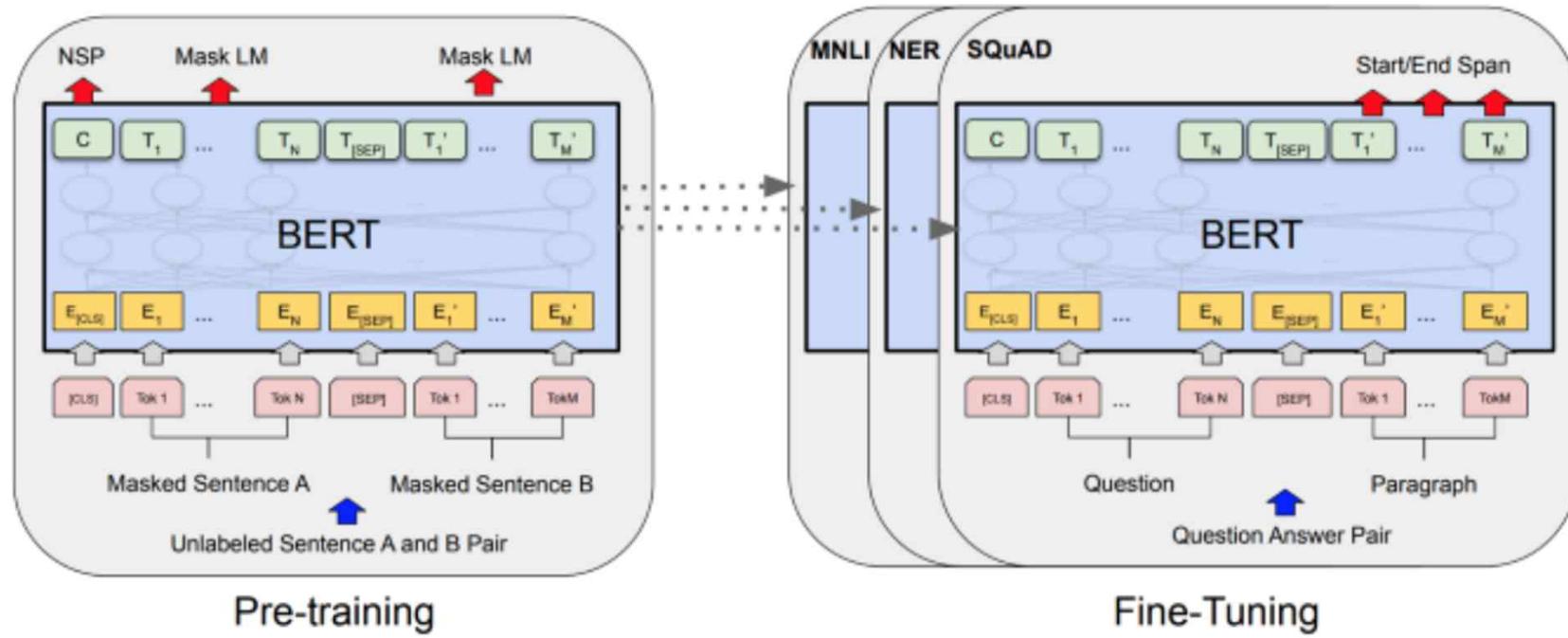
BERT

- 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 task에서 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 task에 적용
 - MNLI(Multi-Genre Natural Language Inference)
 - Named Entity Recognition
 - 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 능력을** 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
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{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, Wikipedia 질의 응답)

Data Example of Question Answering

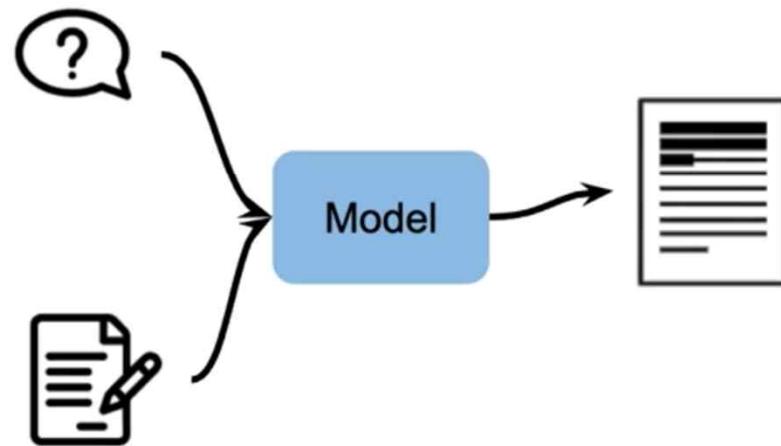
Question: What percentage of the French population today is non - European ?

Context: Since the end of the Second World War , France has become an ethnically diverse country . Today , **approximately five percent** of the French population is non - European and non - white . This does not approach the number of non - white citizens in the United States (roughly 28 – 37 % , depending on how Latinos are classified ; see Demographics of the United States) . Nevertheless , it amounts to at least three million people , and has forced the issues of ethnic diversity onto the French policy agenda . France has developed an approach to dealing with ethnic problems that stands in contrast to that of many advanced , industrialized countries . Unlike the United States , Britain , or even the Netherlands , France maintains a " color - blind " model of public policy . This means that it targets virtually no policies directly at racial or ethnic groups . Instead , it uses geographic or class criteria to address issues of social inequalities . It has , however , developed an extensive anti - racist policy repertoire since the early 1970s . Until recently , French policies focused primarily on issues of hate speech — going much further than their American counterparts — and relatively less on issues of discrimination in jobs , housing , and in provision of goods and services .

Target: Approximately five percent

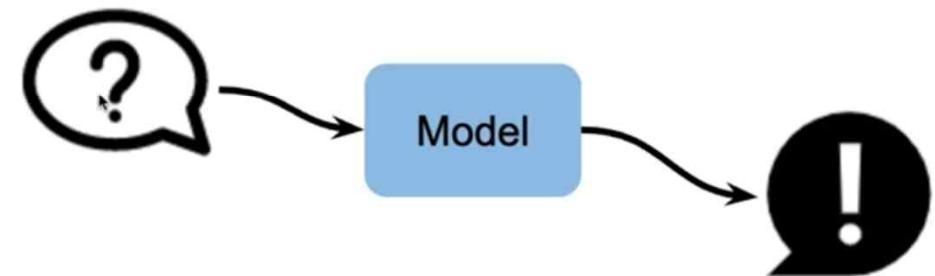
(참고) Question Answering 의 종류

Context-based



context (상황 예문)를 주고 질문을 하면,
model 이 context 내의 정답 위치를 대답

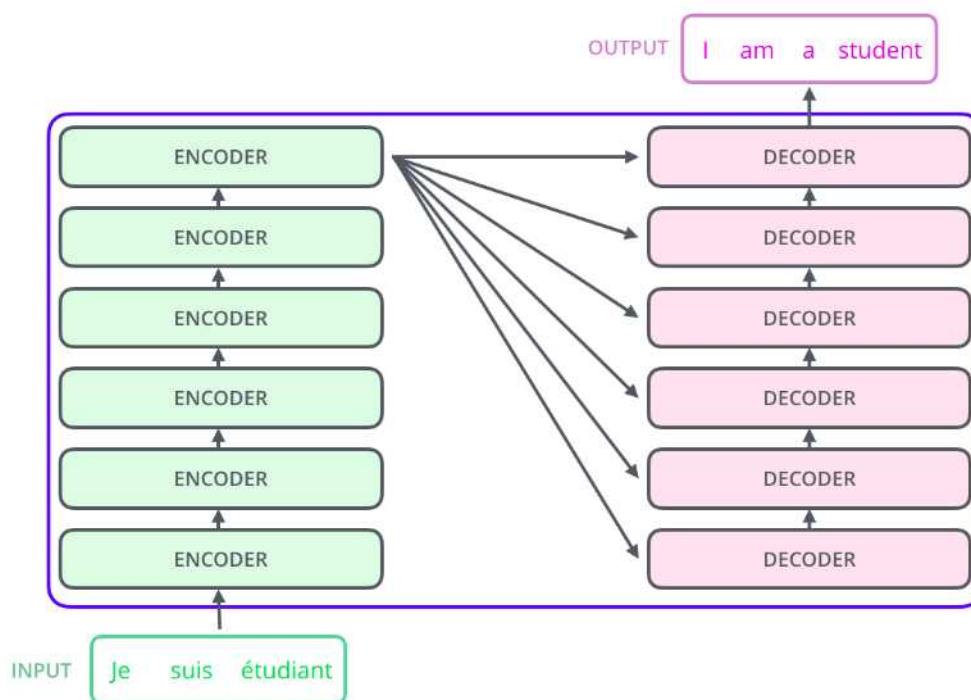
Closed book



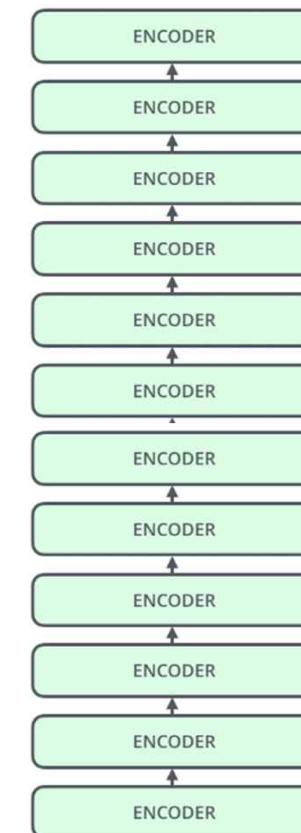
context 나 외부 정보 없이 질문만 하면,
model 이 pre-trained knowledge 를 기초로
답변을 예측

BERT 구성

Transformer
6 encoders + 6 decoders



BERT
12/24 encoders



MLM, Next Sentence Prediction

How to train BERT ?

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

모든 input 은 [CLS]
token 으로 시작

Output token 갯수는
input 과 동일

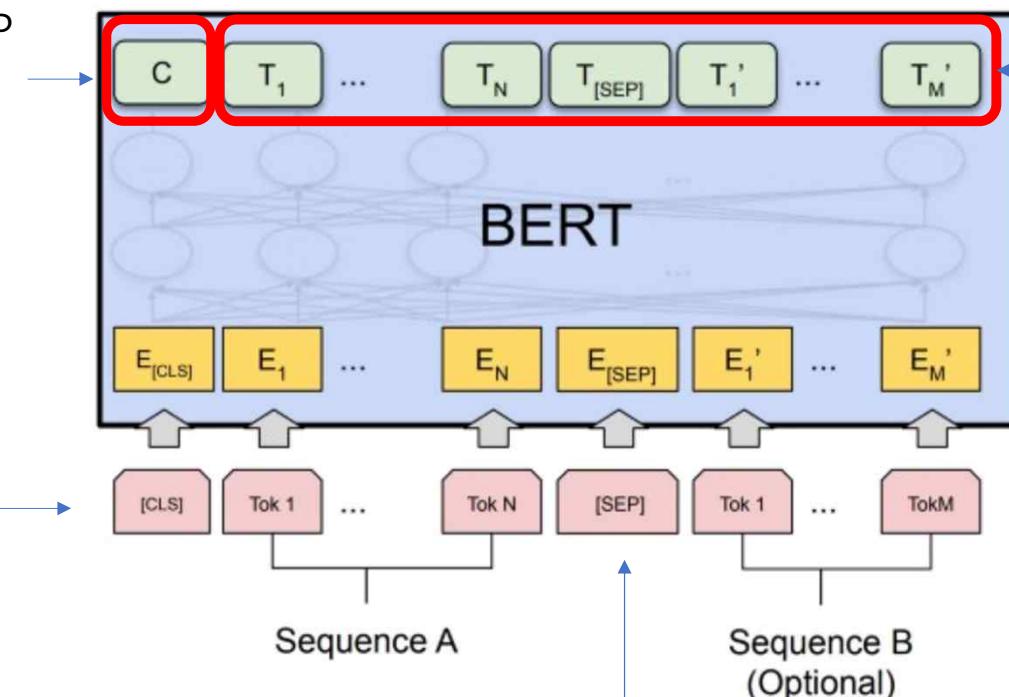
non-classification task 에서
사용

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

→ 2) Next Sentence Prediction

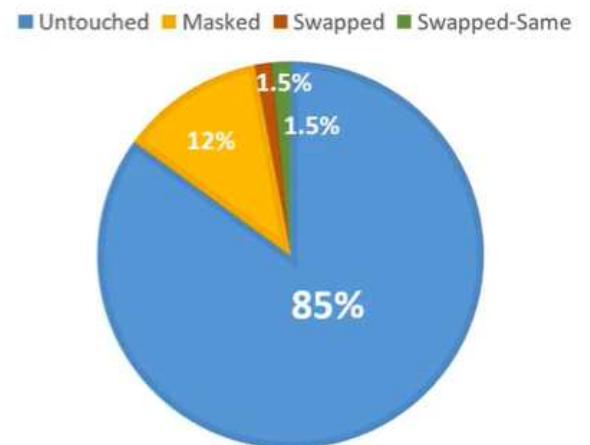
1) Masked Language
Modeling

2가지 Task

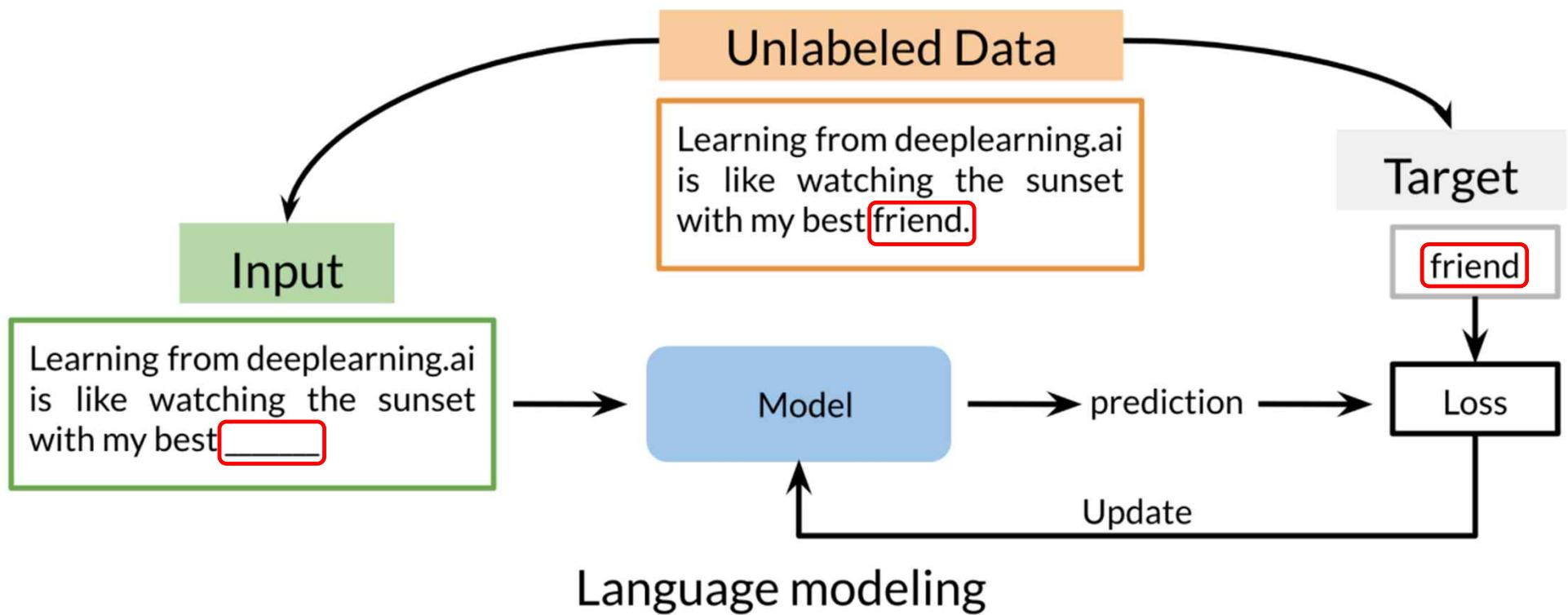


Train 1 : Masked Language Modeling (MLM)

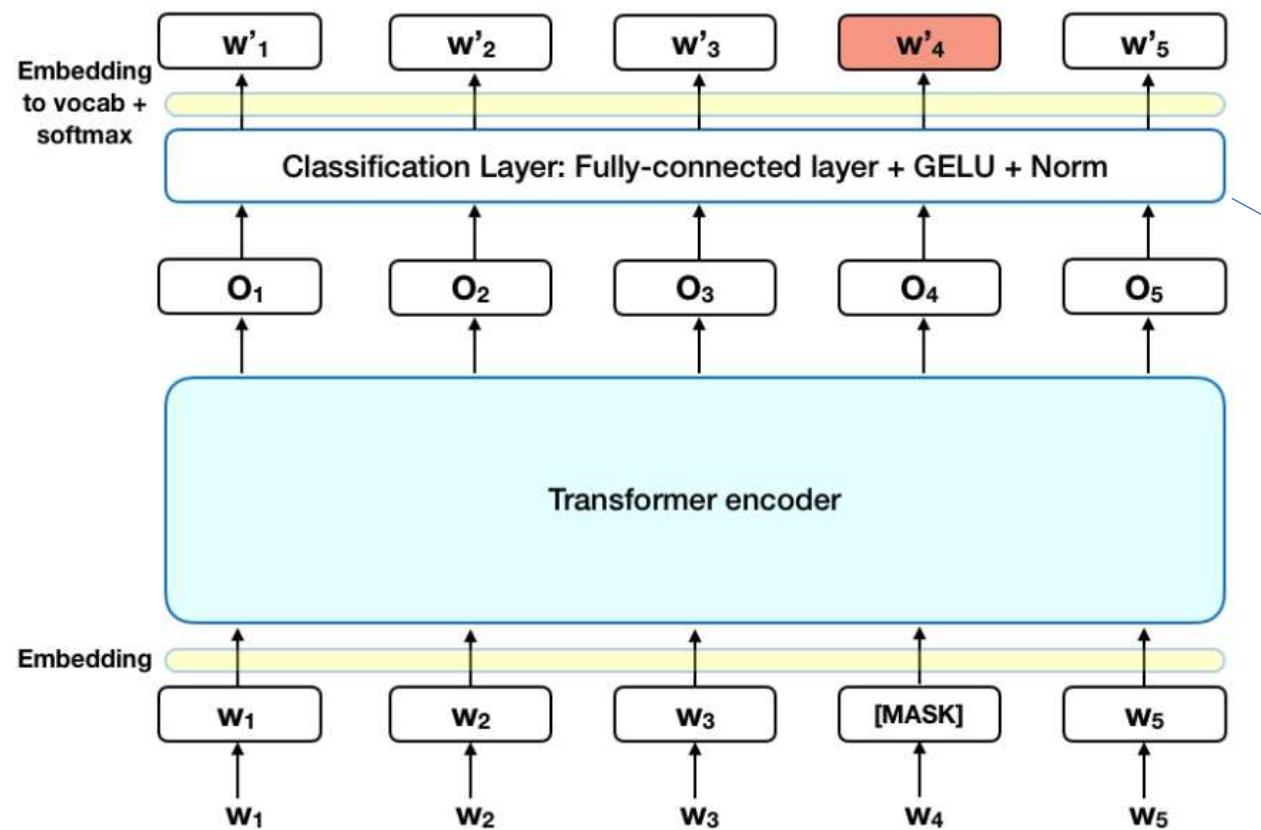
- Input에서 random하게 몇 개의 token을 masking
 - **Input:** the man went to the [MASK] to buy a [MASK] of milk
Label: store gallon
- Left, right context만 가지고 masked word의 원래 단어를 예측
 - 전체 학습 데이터 token의 15%를 masking.
 - 이중 80%는 [MASK] token으로 replace
 - 10%는 random word로 replace
 - 10%는 unchanged (tagging만 바뀐 것으로 표시)



Self-supervised Learning Task



MLM training – 단어 예측



The BERT **loss function**은 masked value 의 prediction 만 고려하고 non-masked word 는 무시

Encoder output 상단에 softmax classification layer 추가
→ Masked word 의 확률분포 예측

Train 2 : Next Sentence Prediction

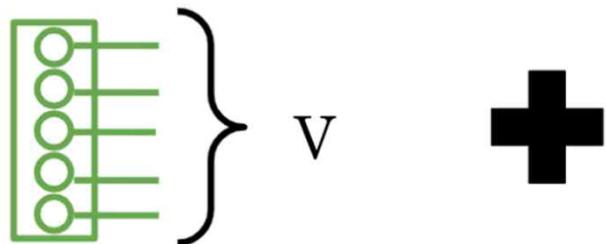
- Training data 로 sentence pair 사용
- 주요 목적은 output 의 C([CLS]) 를 train 시키는 것
- 예를 들어 100,000 개의 sentence 가 있고 BERT language model 을 pre-train 시키려면 50,000 개 pair 의 training data 존재
 - 50% 의 pair 는 second sentence 가 실제 다음 sentence (corpus 에서 연속된 문장)
 - 나머지 50% pair 는 second sentence 를 corpus 에서 random 선택 (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

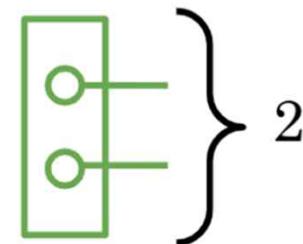
Objective 1:
Multi-Mask LM

Loss: Cross Entropy Loss



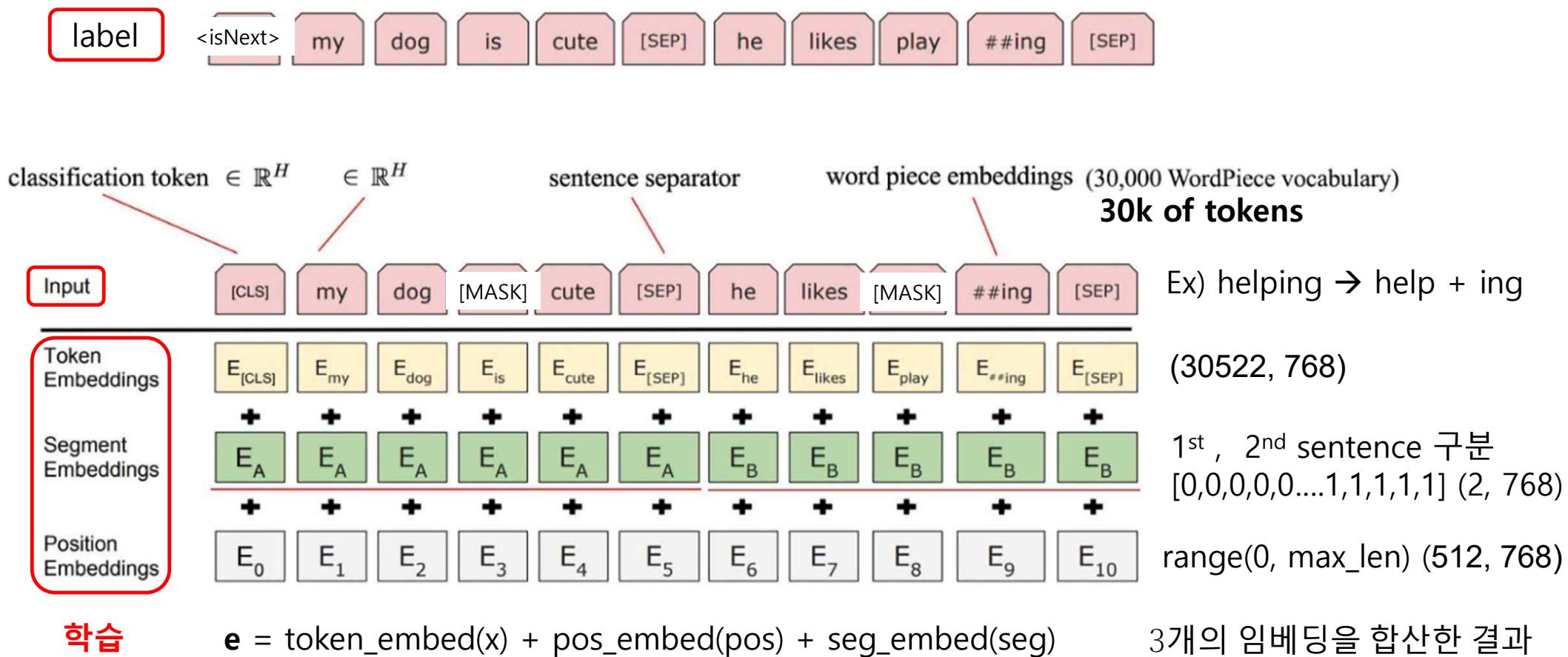
Objective 2:
Next Sentence Prediction

Loss: Binary Loss



두 가지 목적함수를 합산

BERT 의 Input Embeddings – All Together

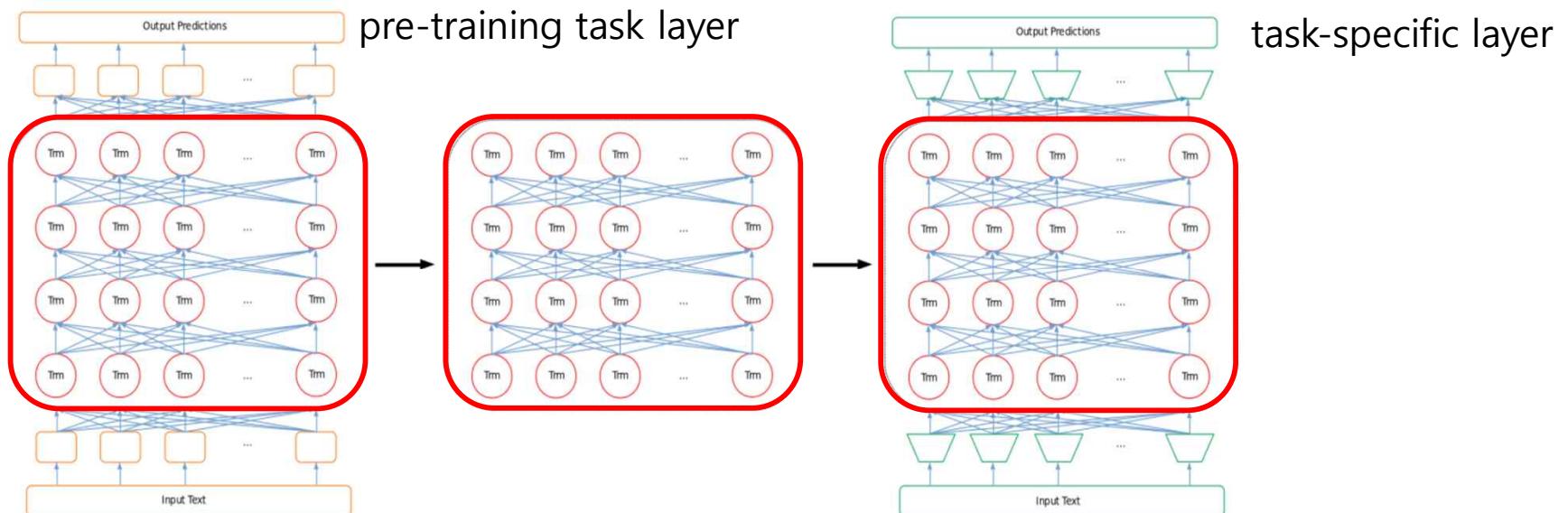


How to Fine-tuning BERT ?

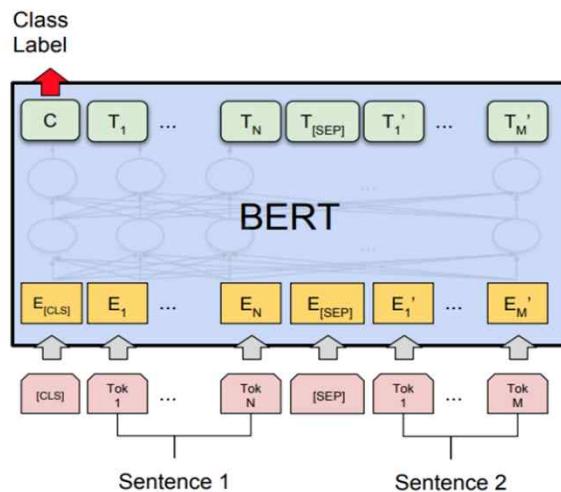
- BERT는 기본적으로 두 문장을 입력으로 취할 수 있도록 설계(두 번째 문장의 입력은 *optional*) 되었기 때문에 GLUE Dataset에 대해 훈련을 진행할 때에는 입력 구조를 바꿀 필요 없음
 - **QQP** (두개의 질문이 서로 유사한가) – 두 입력 문장을 두 질문으로 대체
 - **CoLA** (문장이 문법적으로 맞는가) – 하나의 문장만 입력
 - **QNLI** (문장 B가 문장 A의 질문에 대한 답을 포함하는지) - 질문과 문단을 각각의 문장 입력 값으로 대체
- 사전 훈련에 사용되었던 분류층(pre-training task layer)을 제거하고, 이를 GLUE 태스크를 수행하기 위한 레이어(task-specific layer)로 변경
- 대부분의 경우 변경해주는 레이어 역시 이진 분류 계층

Fine-tuning process of BERT

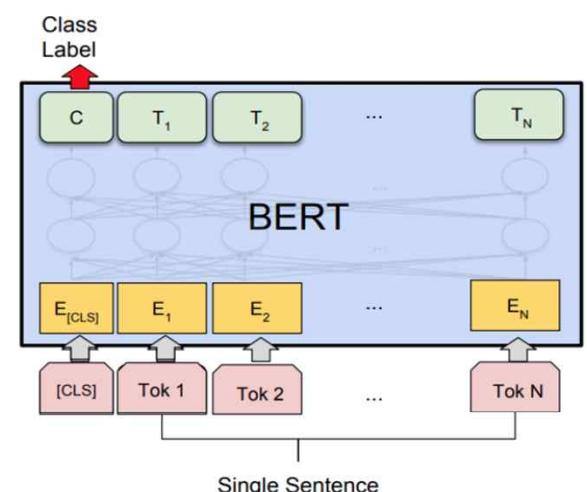
- 위 과정을 거쳐 변형된 모델을 n번의 epoch 동안 재학습(Fine-Tuning)
- 이때 사전학습 모델에서 차용한 모델 중간부는 자연어를 잘 ‘이해’하는 파라미터를 지니고 있기 때문에 우리가 풀고자 하는 문제 해결에 큰 도움이 됨



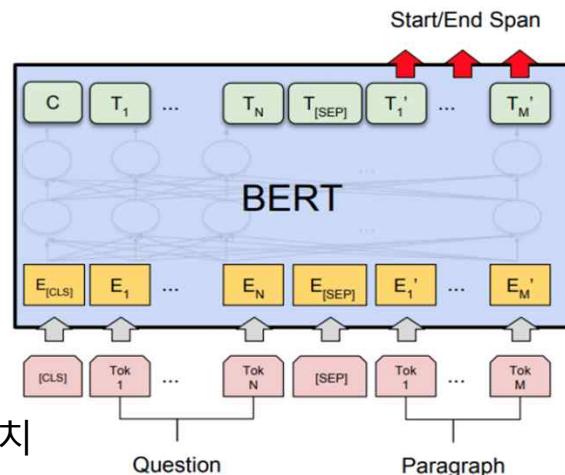
Classification Task



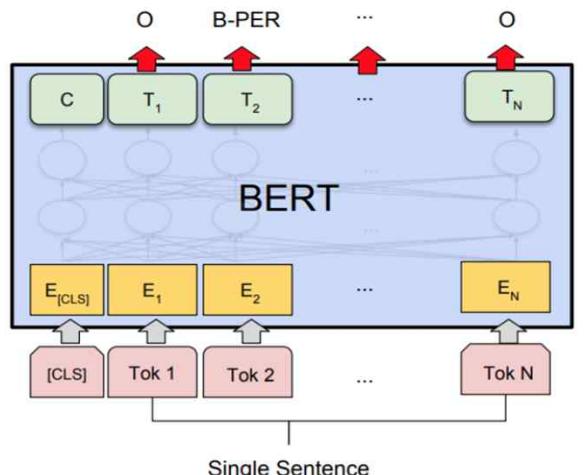
(a) Sentence Pair Classification Tasks:
MNLI, QQP, QNLI, STS-B, MRPC,
RTE, SWAG



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



(c) Question Answering Tasks:
SQuAD v1.1



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

Non-Classification Task

Question – 질문

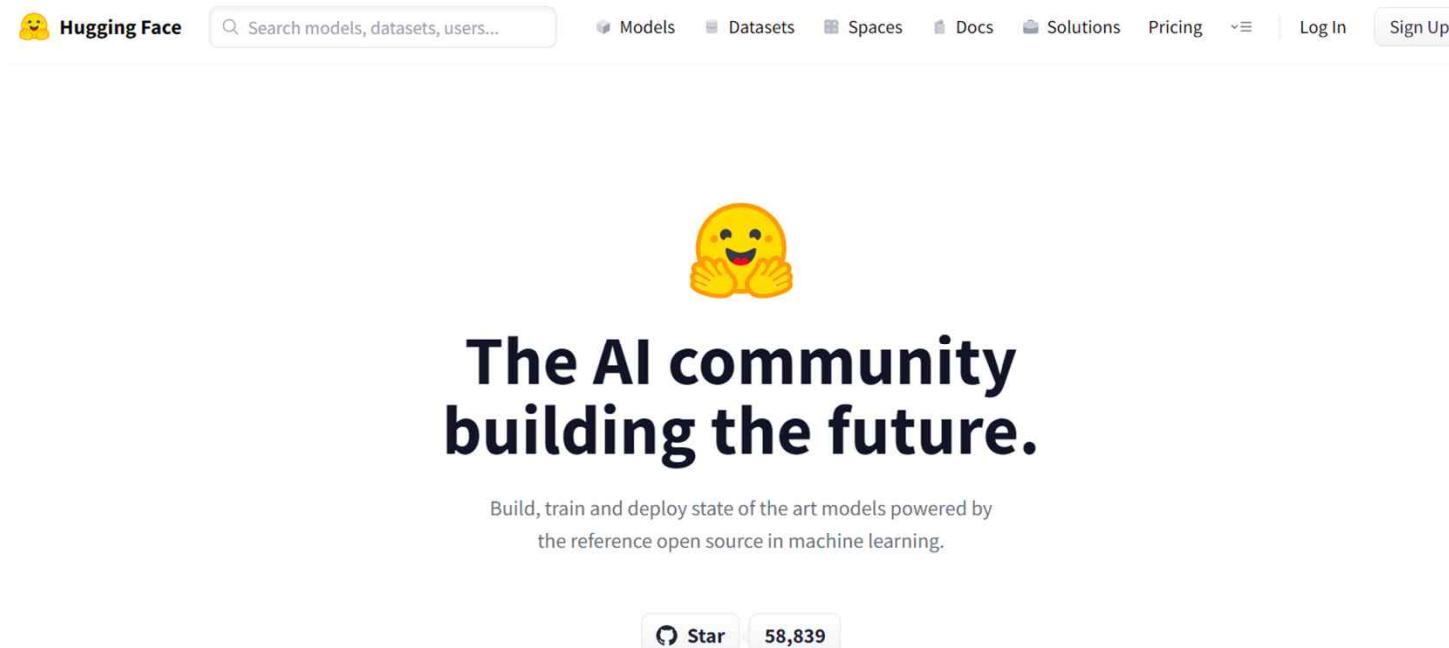
Paragraph – 본문

Start – 문장 중 정답 시작 위치

End – 문장 중 정답 끝 위치

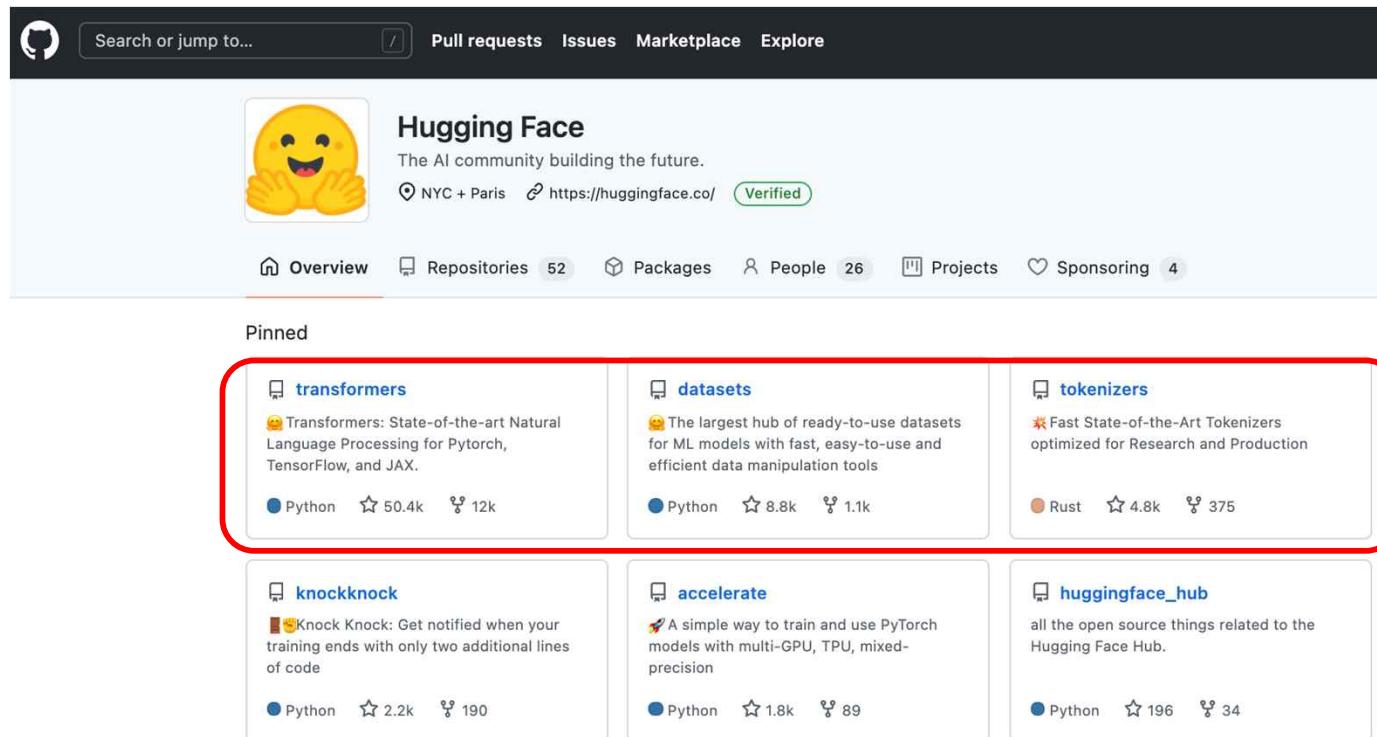
HuggingFace transformers library (<https://huggingface.co/>)

- 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: 300-HuggingFace pipeline quickstart

- HuggingFace Library 사용 (transformers package)
- pipeline 을 이용한 downstream task 실행
 - 감정 분석
 - 텍스트 생성(영문)
 - 이름 개체 인식(NER)
 - 질문 답변
 - 마스킹된 텍스트 채우기
 - 요약
 - 번역
 - 특징 추출

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

wandb

- Weights & Biases의 약자로, 머신러닝 및 딥러닝 프로젝트의 학습 과정을 시각화·추적·관리할 수 있게 해주는 도구
- 대표적으로 Hugging Face Trainer, PyTorch, TensorFlow 등과 쉽게 연동

```
17 # 모델 학습 수행
18 trainer.train()

...
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the pac
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP downl
model.safetensors: 100% 714M/714M [00:02<00:00, 379MB/s]

Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-multilingual-cased and are newly initialized: ['cl
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

wandb: WARNING The 'run_name' is currently set to the same value as 'TrainingArguments.output_dir'. If this was not intended, please specify a different
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize?ref=models
wandb: Paste an API key from your profile and hit enter:
```

API KEY 입력

Copy this key and paste it into your command line to authorize it.

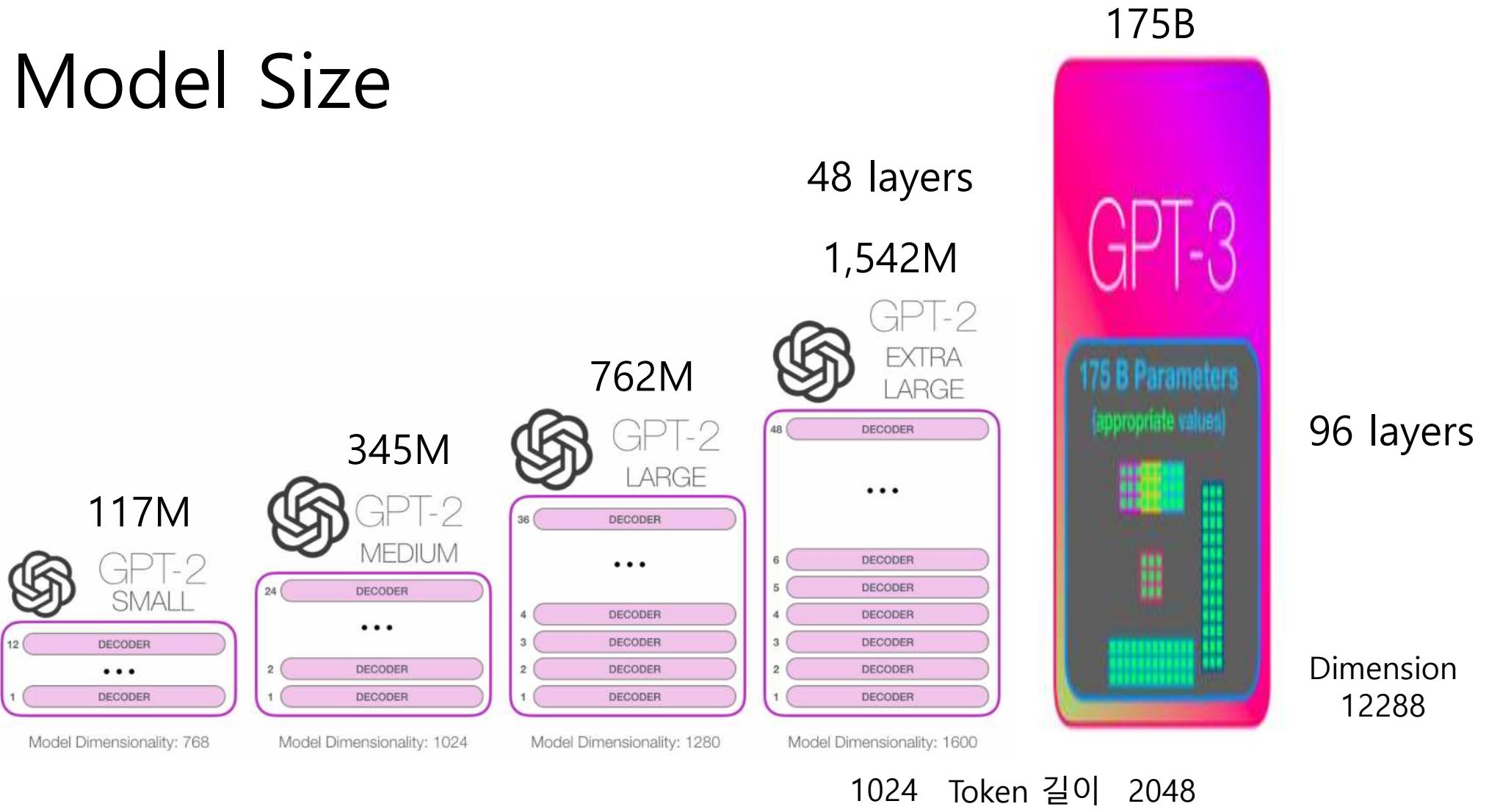
732c979ca7... ☐

GPT-3

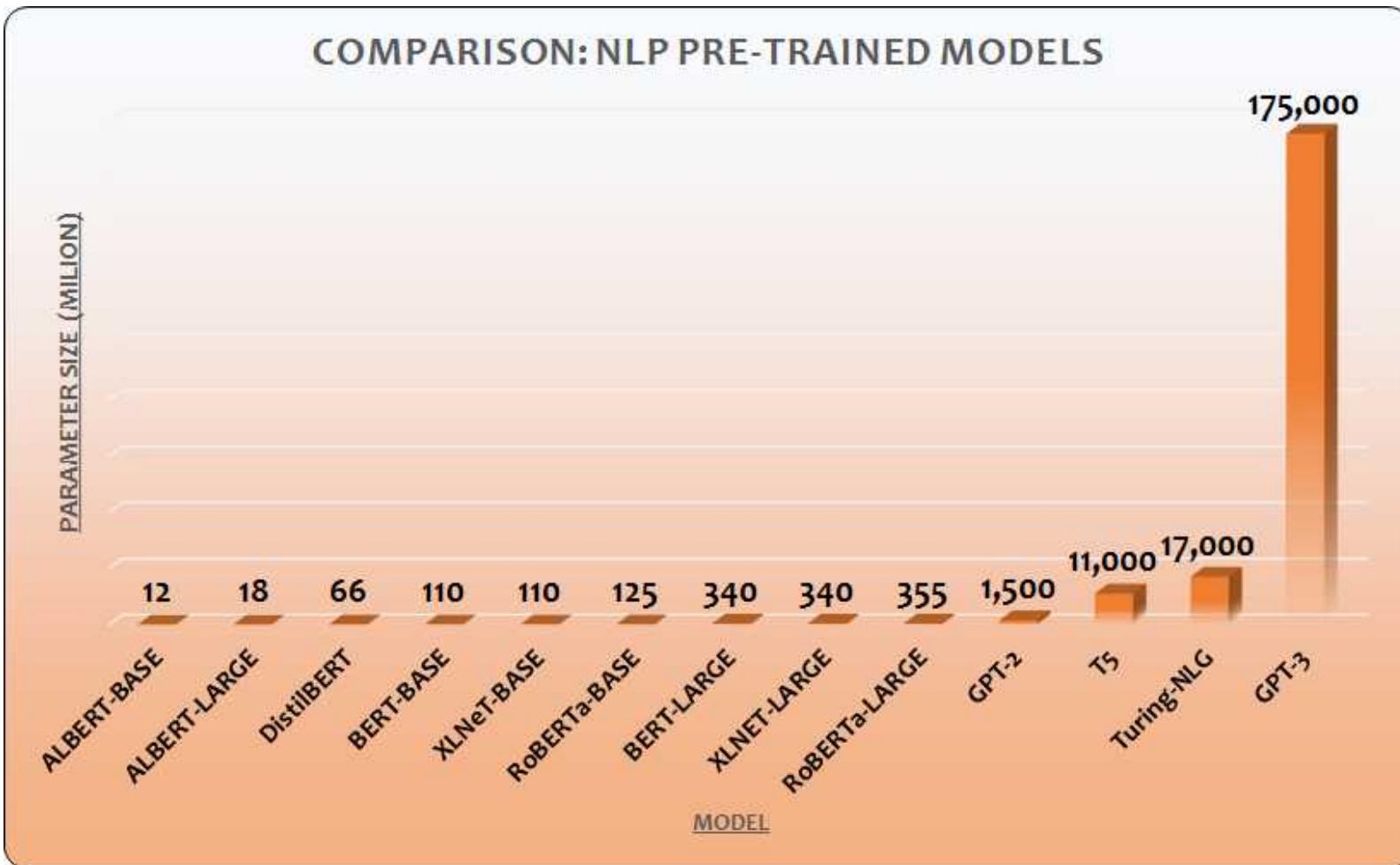
What is GPT-3 ?

- 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 billion에서 175 billion으로 확장
 - GPT-3 가 생성한 가짜 기사 구분 by human – 52% (인간은 구분 못함)

Model Size

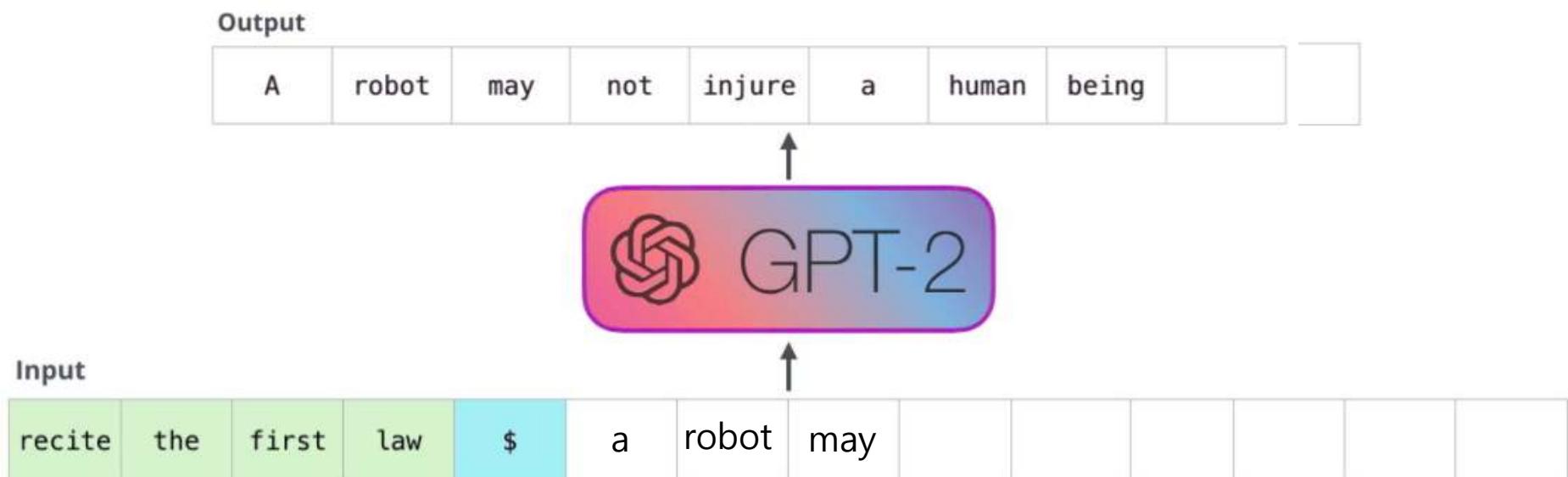


GPT-3 : parameter size – 175 Billions

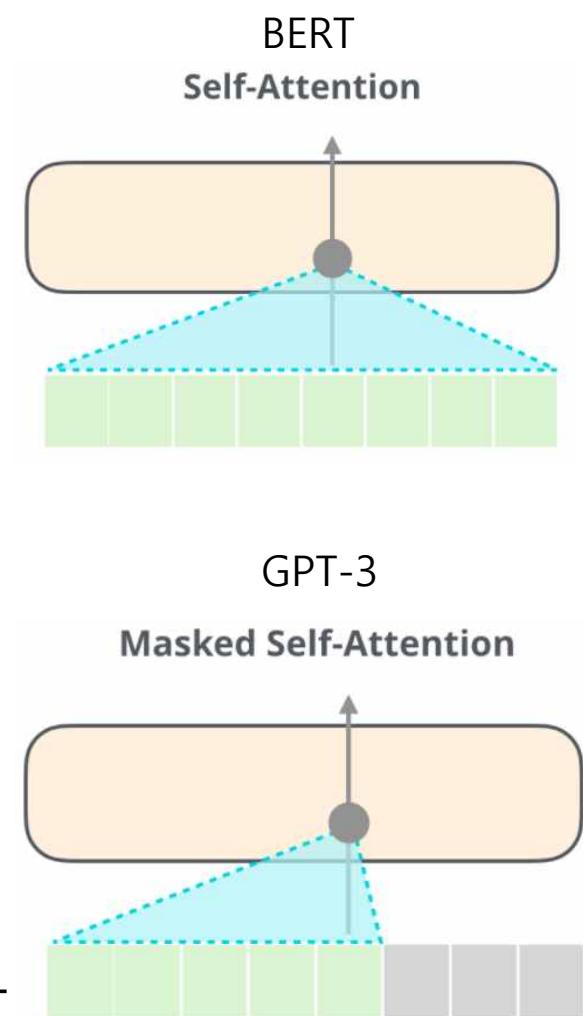
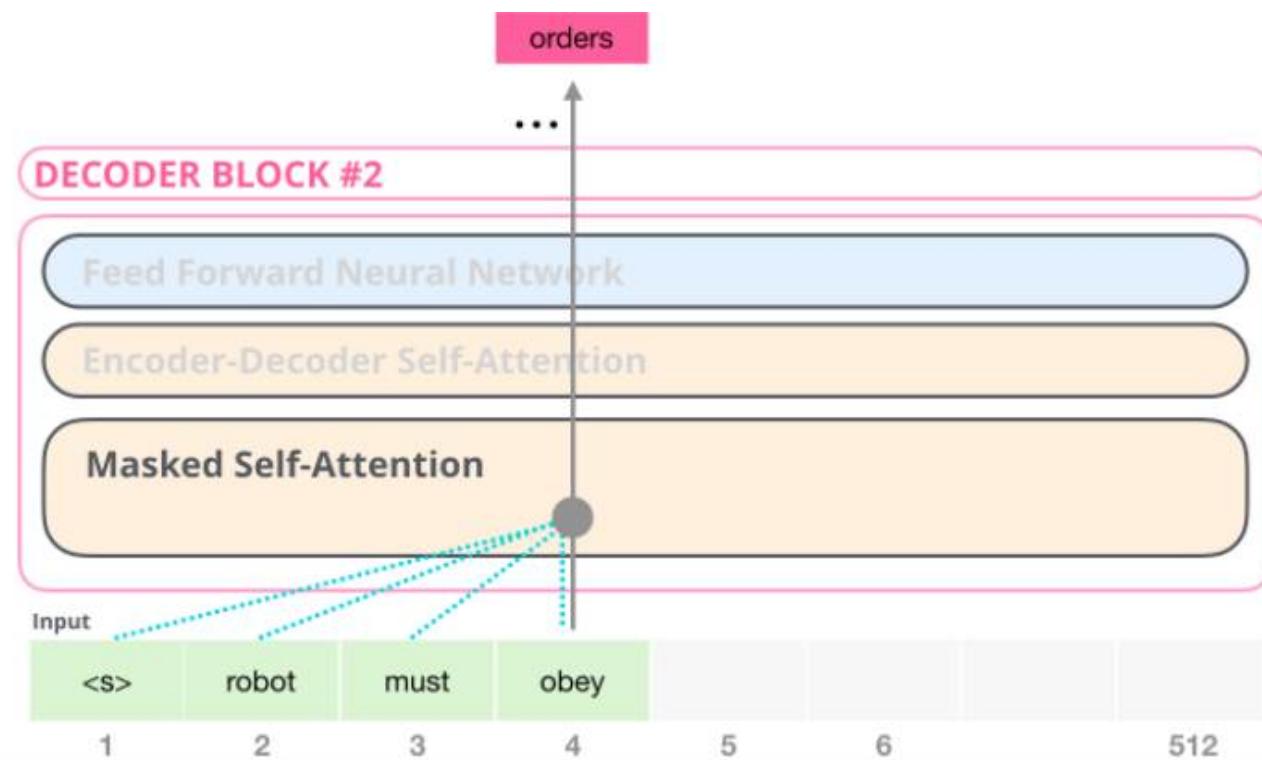


GPT 는 auto-regressive 한 입력 구조

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

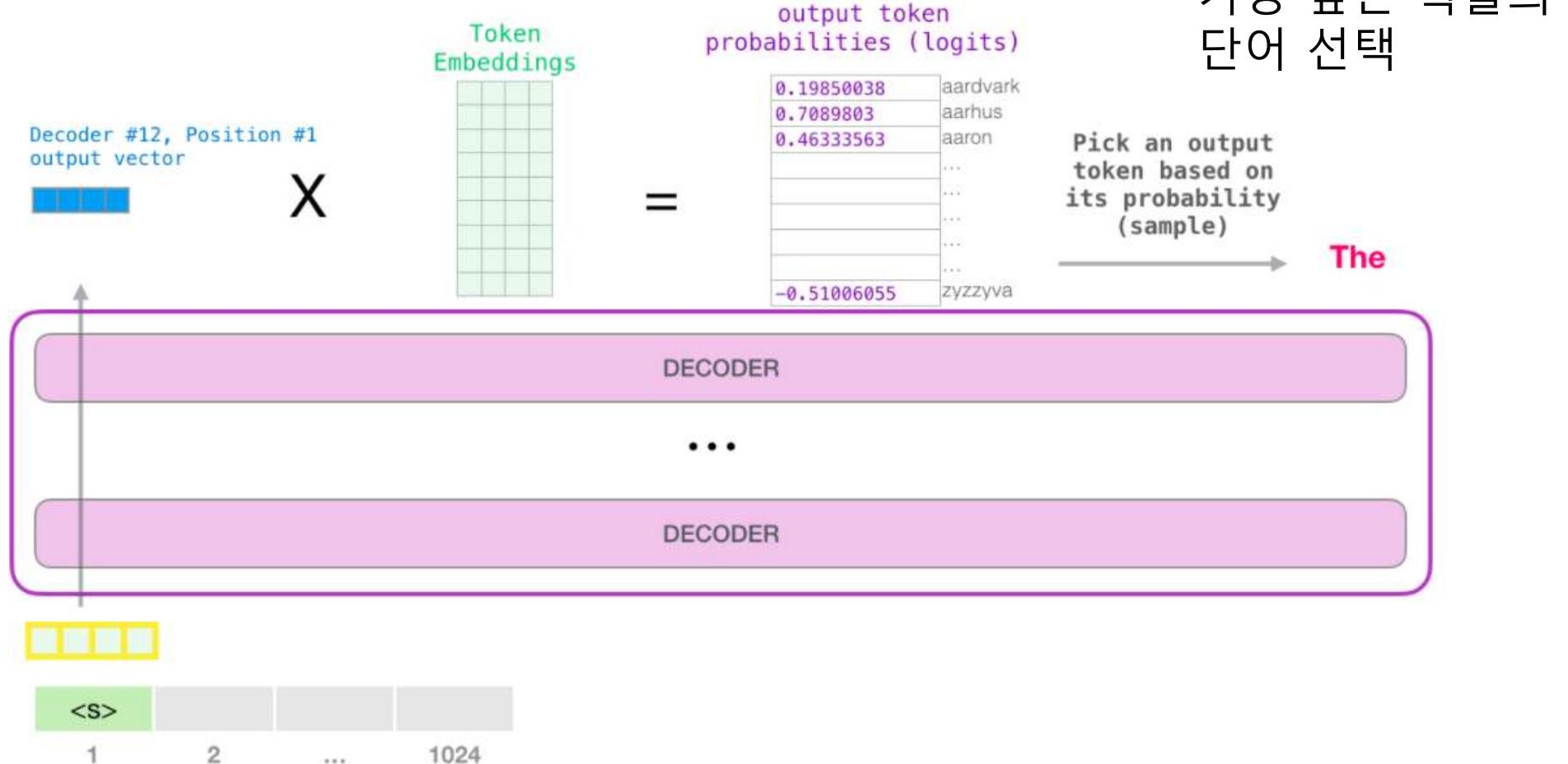


Masked Self-Attention



Next token 예측을 위해 현재 및 이전 토큰만 참조 가능

Output token probability



GPT-3 : Few shot learning

- 지도학습
 - 말, 얼룩말, 젖소 를 구분하기 위해서는 다양한 사진 필요
→ CNN model 의 학습 방법
- Zero shot learning
 - 말과 젖소의 사진만으로 학습 → “얼룩말은 젖소의 색을 가진 말”
 - Model 은 얼룩말 사진을 구분
- One shot learning
 - 한장의 원숭이 사진을 학습
 - Model 은 다른 원숭이 사진을 구분
- Few shot learning
 - 몇장의 강아지 사진을 보여줌
 - Model 은 다른 강아지 사진을 구분



GPT-3

GPT-3 활용 예 - 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 [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;"))
```

prompt = "Display the lowest salary from the Worker table."

'output: Select min(Salary) from Worker;\n'

prompt = "Tell me the count of employees working in the department HR."

"output: SELECT COUNT(*) FROM worker WHERE DEPARTMENT = 'HR';\n"

prompt = "Get salary details of the Workers whose AGE lies between 25 and 35"

output: Select Salary from Worker where AGE between 25 and 35;

GPT-3 활용 예 – 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 활용 예 – 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 돋기 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.

GPT-3 활용 예 - 문장 이해

최근에 발견된 구크럭스라는 행성에서 다양한 과일이 새롭게 발견되었습니다. ‘네오스키즐’이라는 과일은 자주색이고 캔디맛이 난다. ‘로헥클’은 회색 빛 파란 과일인데 새콤하고 약간 레몬 같다. ‘파우니츠’는 밝은 녹색이고 달콤하기 보다는 짭짤하다. 이 행성에는 ‘루프노바스’가 많은데 이건 야광 핑크 느낌에 솜사탕맛이 난다. 마지막으로 ‘그로울스’라는 과일이 있다. 이건 매우 시고 쓴 맛이 나며, 창백한 오렌지색 빛이 돈다.

구크럭스 행성의 과일을 요약한 표를 만들어보세요:

- | 과일 | 색깔 | 맛 |
- | 네오스키즐 | 자주색 | 달콤 |
- | 로헥클 | 회색빛 파란색 | 새콤 |
- | 파우니츠 | 밝은 녹색 | 짭짤 |
- | 루프노바스 | 야광 핑크 | 솜사탕맛 |
- | 그로울스 | 창백한 오렌지 | 신 맛 |

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

GPT-3 활용 예 – 의학 지식

질문: 10살짜리 남자 아이가 차가운 공기와 알레르기로 인한 지속적인 호흡 곤란과 기침을 겪고 있다. 여러 테스트를 해본 결과 이 환자는 폐쇄성 질환으로 판명되어 약을 처방 받았다. 다음 수용체 중 이 약이 작용할 가능성이 높은 수용체는 무엇인가?

- A) 무스카린성아세틸콜린수용체
- B) G 단백질 연결 수용체
- C) Era
- D) Erb
- E) A-1 수용체

정답은 B

설명: 이 환자는 호흡성 질환인 천식을 앓고 있습니다. 천식을 치료하는 약은 기관지 확장제입니다. 기관지 확장제는 beta-2 수용체에 작용하며, beta-2 수용체는 G 단백질 수용체의 일부

GPT-3 활용 예

- 앱을 만든 사례
- 검색엔진을 만든 사례
- 기타 연주 악보를 만든 사례
- 사업 아이디어를 만든 사례
- auto-programming 사례
https://beta.openai.com/?app=productivity&example=4_2_0
- Jukebox - generates music! (Transformer 활용 사례)
<https://openai.com/blog/jukebox/>

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 → 실질적으로 개발자 대체

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 output에 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

OpenAI GPT

● 생성 모델의 환각 효과 Hallucination Effect

→ "인공지능(AI) 챗봇에 발생하는 환각(할루시네이션) 현상은 여전한 AI 걸림돌입니다. 발생 이유는 아무도 모릅니다. 그래서 명확한 해결책도 없습니다. 사회에 AI를 안전하게 뿌리내리려면 환각 현상을 어떻게든 해결해야 합니다." – 선다 피차이 (구글 CEO)



문재인 대통령의 미국 방문 때 바이든 대통령과 몇 차례의 회담을 했었나?



문재인 대통령은 2023년 5월 20일부터 22일까지 미국을 방문했습니다. 이 기간 동안 문재인 대통령은 조 바이든 미국 대통령과 총 2차례의 회담을 가졌습니다. 첫 번째 회담은 20일 오후 워싱턴 백악관에서 열렸으며, 두 번째 회담은 22일 오전 워싱턴 국무부에서 열렸습니다. 두 차례의 회담에서 양국 정상은 한미 동맹 강화, 북핵 문제, 경제 협력 등 다양한 분야에 대해 논의했습니다.



윤석열 대통령의 미국 방문 때 바이든 대통령과 몇 차례의 회담을 했었나?

다른 답안 보기 ▾



윤석열 대통령은 2023년 4월 20일부터 22일까지 미국을 방문했습니다. 이 기간 동안 윤석열 대통령은 조 바이든 미국 대통령과 두 차례의 회담을 했습니다. 첫 번째 회담은 4월 20일 백악관에서 열렸고, 두 번째 회담은 4월 21일 백악관에서 열렸습니다. 두 차례의 회담에서 윤석열 대통령과 조 바이든 대통령은 한미동맹 강화, 북한 핵 문제, 경제 안보 등 다양한 분야에 대해 논의했습니다.

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

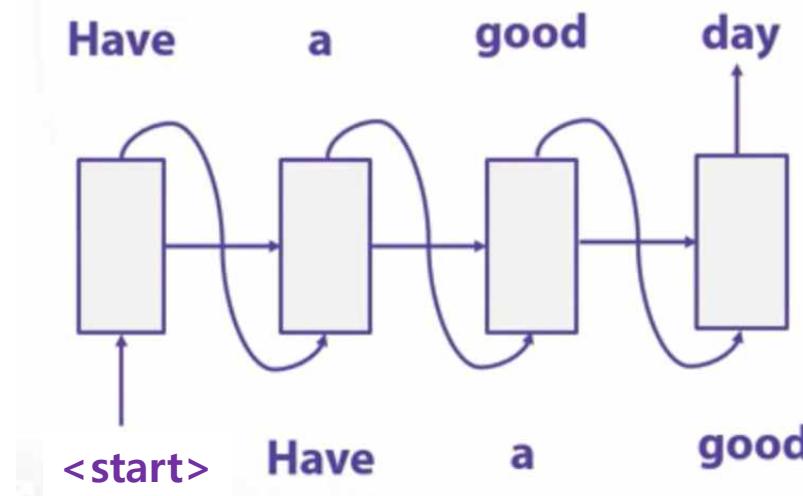
모델이 확률적으로 다음 단어를 예측하기 때문에, 잘못된 단어를 선택할 가능성이 있다.

→ 환각 문제는 아직 해결되지 않았으며 AI 모델 활용의 가장 큰 도전 과제

Language Model

○ 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



Language Model

● Decoding Strategy

→ Greedy 전략

▶ softmax 분포 중 가장 높은 확률 (argmax) 선택, 언제나 같은 번역

▶ argmax

$$P(\text{Over } \boxed{\text{the}} \boxed{\text{line}} \boxed{!} \mid \boxed{\text{Çizgiyi}} \boxed{\text{geçtin}} \boxed{!})$$

→ Sampling 전략 (사후확률분포)

▶ 분포 확률에 따라 Random sampling, 매번 번역이 바뀔 수 있음

▶ `np.random.choice(len(probs), p=probs)`

→ Beam-search 전략

▶ 단순히 첫번째 단어를 argmax로 선택하면 1스텝에서라도 문법 상 실수를 할 경우, 전체 문장의 번역에 큰 실수가 되므로, 각각의 타임스텝 t마다 b 개의 sequence 후보군을 유지

실습 : 150_autoregressive_language_generation

- Autoregressive 문장 생성
- Huggingface의 transformers library 사용
- Pre-trained GPT-2 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
```

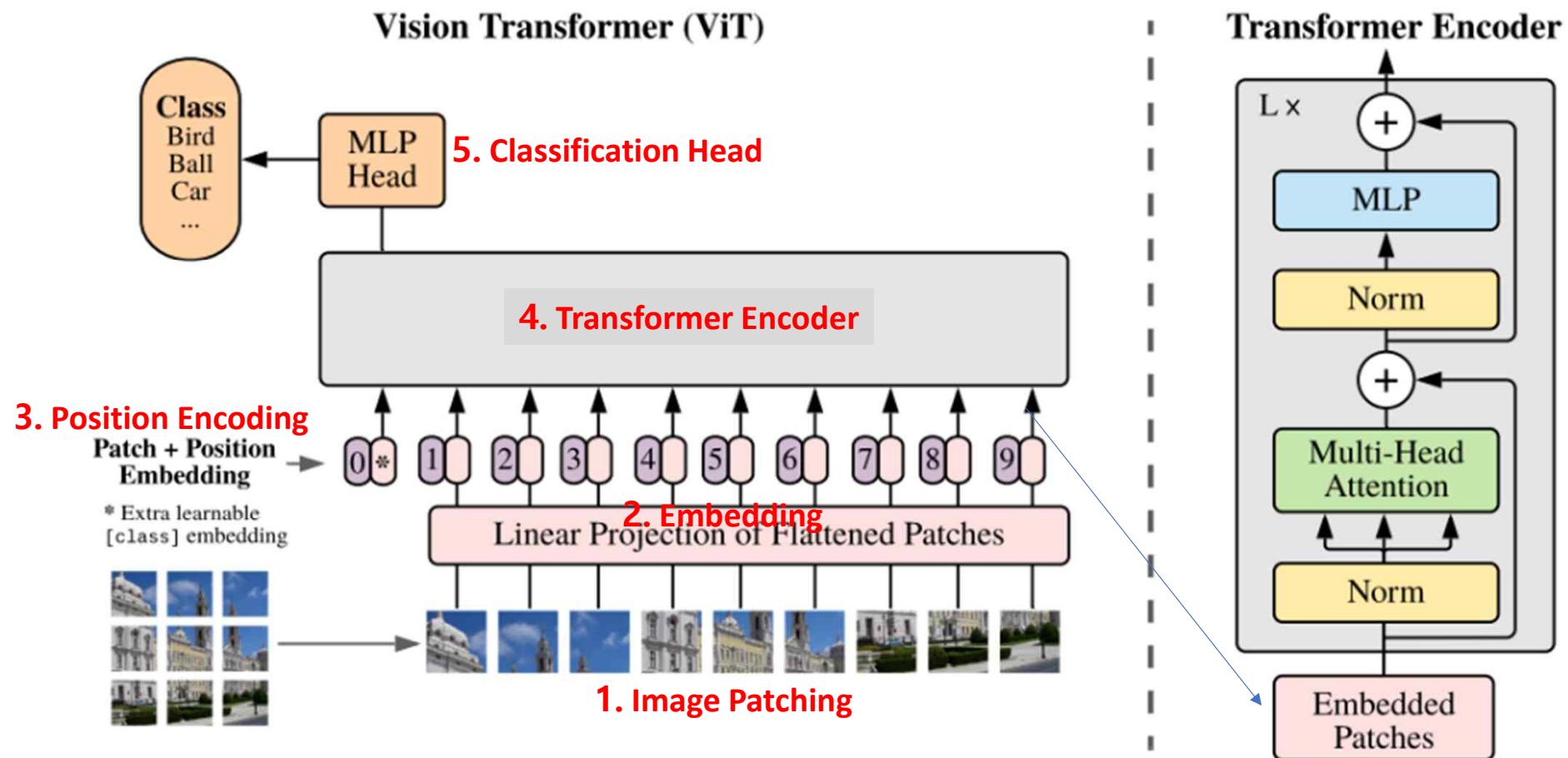
```
tensor([[7454, 2402, 257, 640]])
```

```
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
```

Vision Transformer

Model Overview

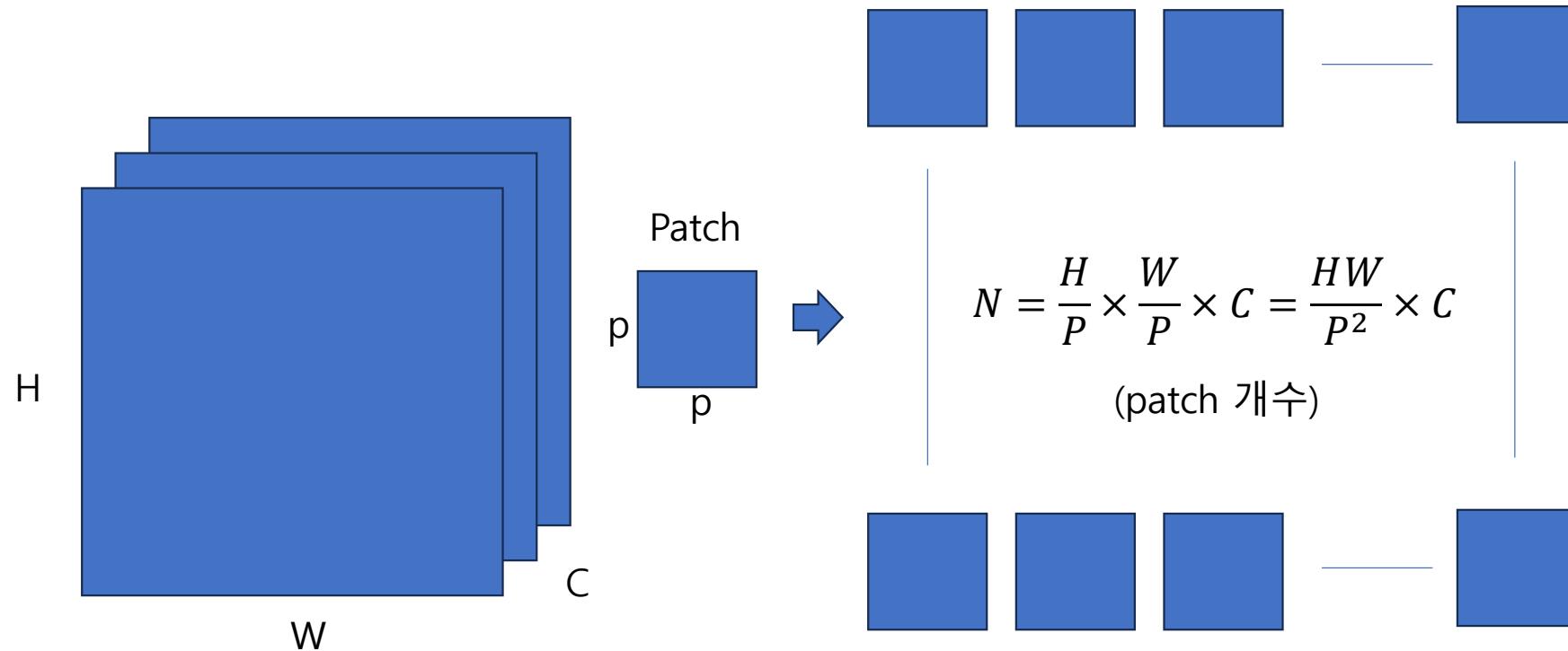
Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M



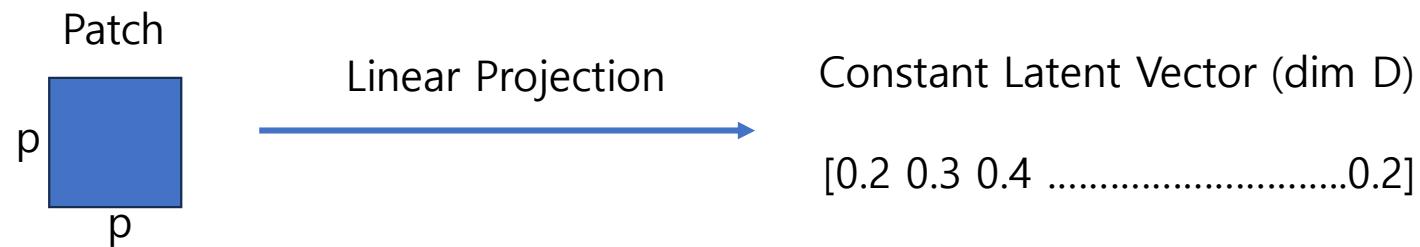
ViT의 작동 방식

1. **Image Patching:** 주어진 이미지를 여러 개의 고정 크기의 패치(patch)로 나눕니다. 예를 들어, 224×224 크기의 이미지를 16×16 크기의 패치로 나누면 총 196개의 패치가 생깁니다.
2. **Embedding:** 각 패치를 1D 벡터로 평탄화(flatten)한 후, 선형 임베딩(linear embedding) 레이어를 통과시켜 고정된 크기의 임베딩 벡터로 변환
3. **Position Encoding:** 각 패치의 위치 정보를 주입하기 위해 위치 인코딩(position encoding)을 추가. 이는 NLP에서의 Transformer와 동일한 개념.
4. **Transformer Encoder:** 앞서 생성한 임베딩 벡터들은 Transformer 인코더에 입력됩니다. Transformer 인코더는 여러 개의 attention 및 feed-forward 네트워크 레이어로 구성.
5. **Classification Head:** Transformer의 최종 출력은 분류 작업을 위한 MLP 레이어(또는 다른 헤더)에 입력

입력 Patch 구성



Linear Projection of Patch



CLIP

(Contrasive Language-Image Pre-training)

Learning Transferable Visual Models From Natural Language
Supervision

사용 dataset

- MS-COCO, Visual Genome: 고품질/소규모 데이터 (100,000개)
- YFCC100M (100M) → filtering 후 고품질 15M개 사용
- 인터넷에 공개되어 있는 400M개의 (image, text) pair로 새로운 dataset 구성
- 텍스트 query는 위키피디아에서 100번 이상 등장하는 단어 50만개로 구축
- 데이터 균형을 위해 query당 이미지-텍스트 pair 개수를 최대 2만개로 제한
- 위 3가지 데이터로 contrastive learning (대조학습) 수행

* constrasive learning 은 서로 다른 두 데이터 포인트가 얼마나 다른지 (또는 유사한지)를 비교하거나 측정하여 유사한 데이터 포인트들이 가까이 있도록 임베딩 공간을 구성

Data example

Google a black dog with white spots

Dreamstime.com
Black Dog with White Spots...

Quora
What are some dog breeds that ar...

Daily Mail
Rowdy the Labrador's vitl...

Can Stock Photo
Puppy black and white with spots. C...

Adobe Stock
Black dog with white spots on the legs, chest ...

PetHelpful
90+ Unique Names for Dogs With Spot...

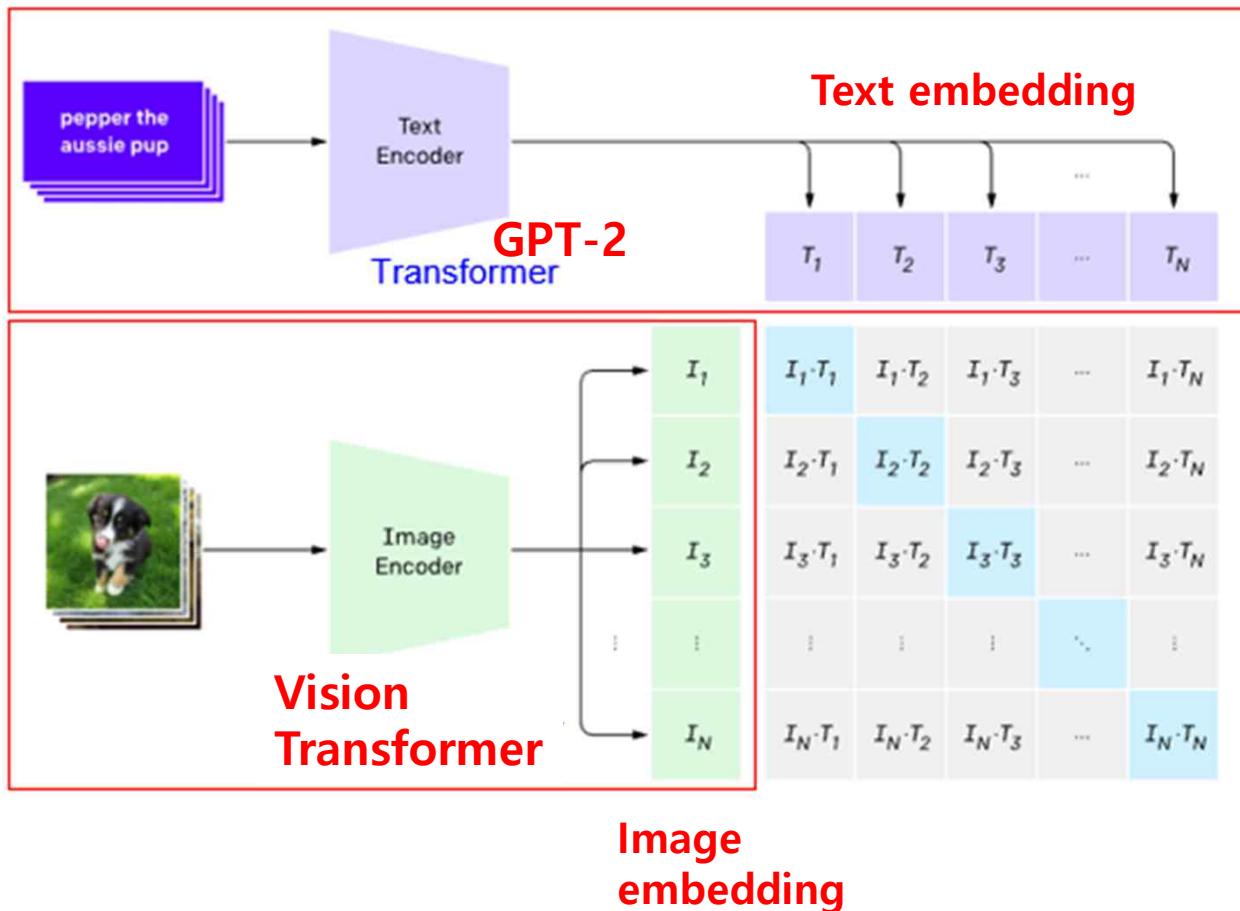
Alamy
Black dog with white spots on the legs...

Reddit
Black Dalmatian With White S...

Shutterstock
Black Dog White Spots Looks Yell...

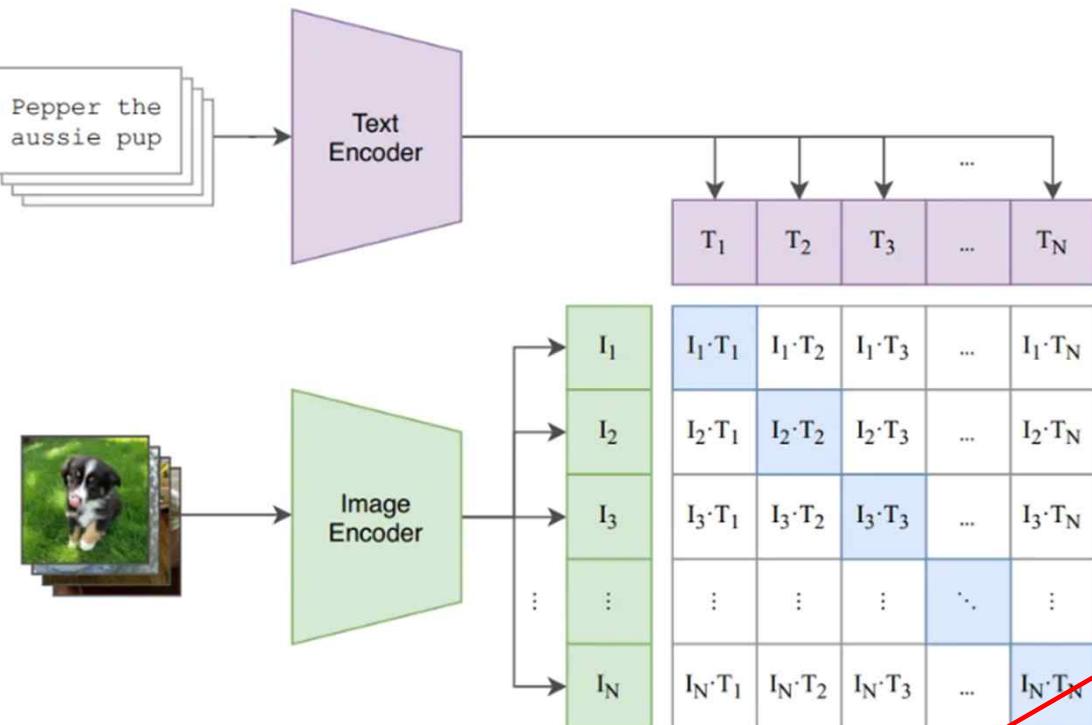
Reddit
My all-black dog develope...

Contrastive Pre-Training



- N개의 이미지, 텍스트를 각각 encoder를 통과시켜 embedding vector 계산 (N: batch 사이즈)
- NxN matrix의 대각선 부분이 positive pair, 그 외 부분은 모두 negative pair
(N개의 positive pair, N^2-N 개의 negative pair)
- Positive pair의 cosine 유사도는 maximize 하고, negative pair의 cosine 유사도는 minimize 하도록 train

(1) Contrastive pre-training



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

```

# image_encoder - ResNet or Vision Transformer
# text_encoder - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, 1]         - minibatch of aligned texts
# W_i[d_i, d_e]  - learned proj of image to embed
# W_t[d_t, d_e]  - learned proj of text to embed
# t               - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T) # [n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

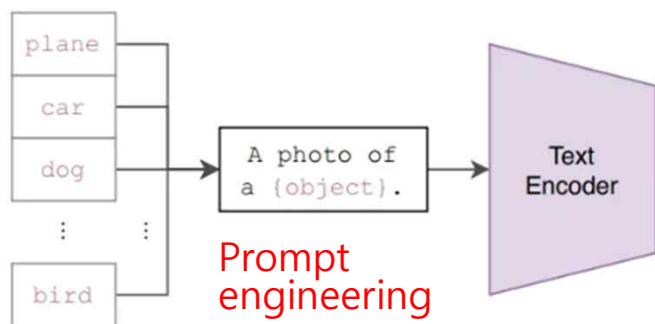
# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)
temperature
# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss  = (loss_i + loss_t)/2

```

이미지와 텍스트에 대한 cross-entropy loss 계산
이미지, 텍스트에 대한 loss를 평균하여 최종 loss 도출

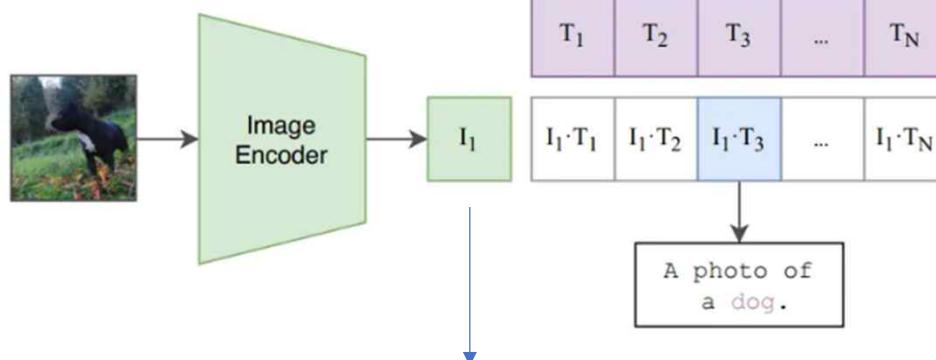
Inference (Zero-shot prediction)

(2) Create dataset classifier from label text



(1)에서 학습된 CLIP 모델을 Fine Tuning 과정 없이 바로 classification에 사용(zero-shot)

(3) Use for zero-shot prediction

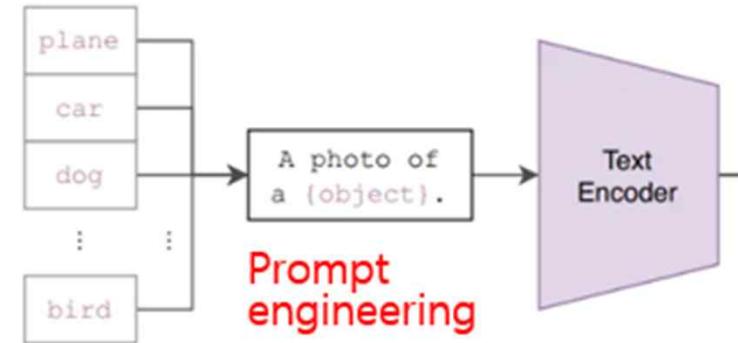


- 분류하고자 하는 클래스 label들에 대해 prompt engineering 수행
- 텍스트 encoder로부터 text embedding 추출
- 가장 유사도(dot product)가 높은 것이 prediction

- 분류하려는 image의 representation

Prompt Engineering

- label → A photo of a {label}
- 동음이의어 문제 해결
- 인터넷에서 수집한 training set의 text는 대체로 full sentence 임
- Oxford-IIIT Pets dataset의 경우 모든 image 가 애완 동물이므로, label → A photo of a {label}, a type of pet
- Ensemble prompting 기법
label → A photo of a big {label} + A photo of a small {label}



MNIST

correct label: 7



correct rank: 1/10 correct probability: 85.32%

a photo of the number: "7".

a photo of the number: "2".

a photo of the number: "1".

a photo of the number: "6".

a photo of the number: "4".

0 20 40 60 80 100

FGVC Aircraft

correct label: Boeing 717



correct rank: 2/100 correct probability: 9.91%

a photo of a mcdonnell douglas md-90, a type of aircraft.

a photo of a boeing 717, a type of aircraft.

a photo of a fokker 100, a type of aircraft.

a photo of a mcdonnell douglas dc-9-30, a type of aircraft.

a photo of a boeing 727-200, a type of aircraft.

0 20 40 60 80 100

German Traffic Sign Recognition Benchmark (GTSRB)

correct label: red and white triangle with exclamation mark warning



correct rank: 1/43 correct probability: 45.75%

a zoomed in photo of a "red and white triangle with exclamation mark warning" traffic sign.

a zoomed in photo of a "red and white triangle with black right curve approaching warning" traffic sign.

a zoomed in photo of a "red and white triangle car slidding / slipping warning" traffic sign.

a zoomed in photo of a "red and white triangle rough / bumpy road warning" traffic sign.

a zoomed in photo of a "red and white triangle with black left curve approaching warning" traffic sign.

0 20 40 60 80 100

EuroSAT

correct label: annual crop land



correct rank: 4/10 correct probability: 12.90%

a centered satellite photo of permanent crop land.

a centered satellite photo of pasture land.

a centered satellite photo of highway or road.

a centered satellite photo of annual crop land.

a centered satellite photo of brushland or shrubland.

0 20 40 60 80 100

Training details

- Image encoder
 - ResNet50 series (ResNet-50, ResNet-101, RN50x4, RN50x16, RN50x64)
 - Vision Transformer (ViT-B/32, a ViT-B/16, ViT-L/14)
- Text encoder – Transformer
- Very large minibatch size – 32,768
- 다양한 학습 및 메모리 최적화 기법 사용
- 가장 큰 모델 기준 592개의 v100 GPU 이용 18일 소요

실습: 910_Interacting_with_CLIP

- CLIP model을 download 받아 실행
- 임의의 이미지와 텍스트 간의 유사성 계산
- Zero shot 이미지 분류 수행 방법 이해

camera.png
a person looking at a camera on a tripod



rocket.jpg
a rocket standing on a launchpad



chelsea.png
a facial photo of a tabby cat



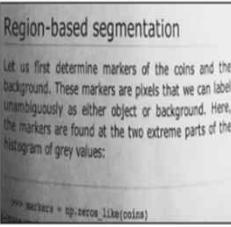
coffee.png
a cup of coffee on a saucer



astronaut.png
a portrait of an astronaut with the American flag



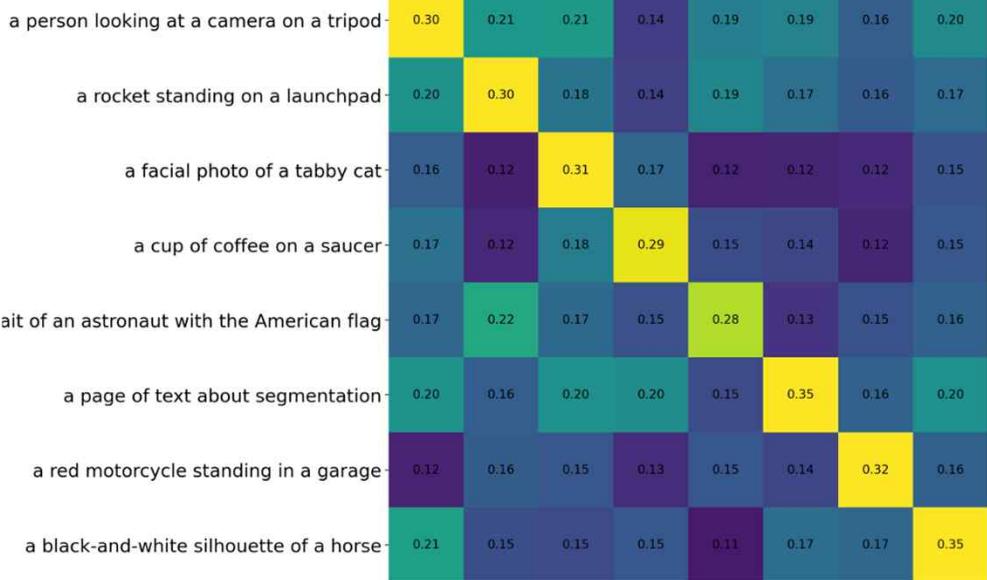
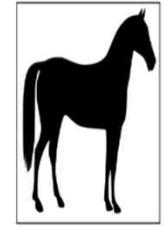
page.png
a page of text about segmentation



motorcycle_right.png
a red motorcycle standing in a garage



horse.png
a black-and-white silhouette of a horse



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 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-4o	2024 OpenAI	- 다중 모달 지원 (텍스트, 이미지, 음성) - 빠른 응답	AI 챗봇(ChatGPT)
Claude 3.5	2024 Anthropic	- 안전성을 강화한 AI 모델 - 강력한 추론 및 논리적 사고 능력	AI 윤리 및 법률 도구 교육 및 연구 보조 데이터 분석
Gemini 1.5	2024 Google	- 멀티모달 처리 지원 - 1백만 토큰 이상의 컨텍스트 윈도우 지원	복잡한 대화형 AI 시스템 다국어 번역 및 문서 생성
Llama 3	2024 Meta	- 오픈소스 LLM	경량화된 AI 모델 연구
o1	2025 OpenAI	- 강화 학습을 통한 고급 추론 능력	학술 논문 분석, 실험 데이터 분석, 새로운 연구 가설 제안 등
DeepSeek	2025 DeepSeek	- 오픈 소스 LLM (MIT 라이선스) - 혼합전문가(Mixture of Experts) 구조와 강화학습으로 모델 개선	개선저비용 고효율 추구

LLM API (Application Programming Interface)

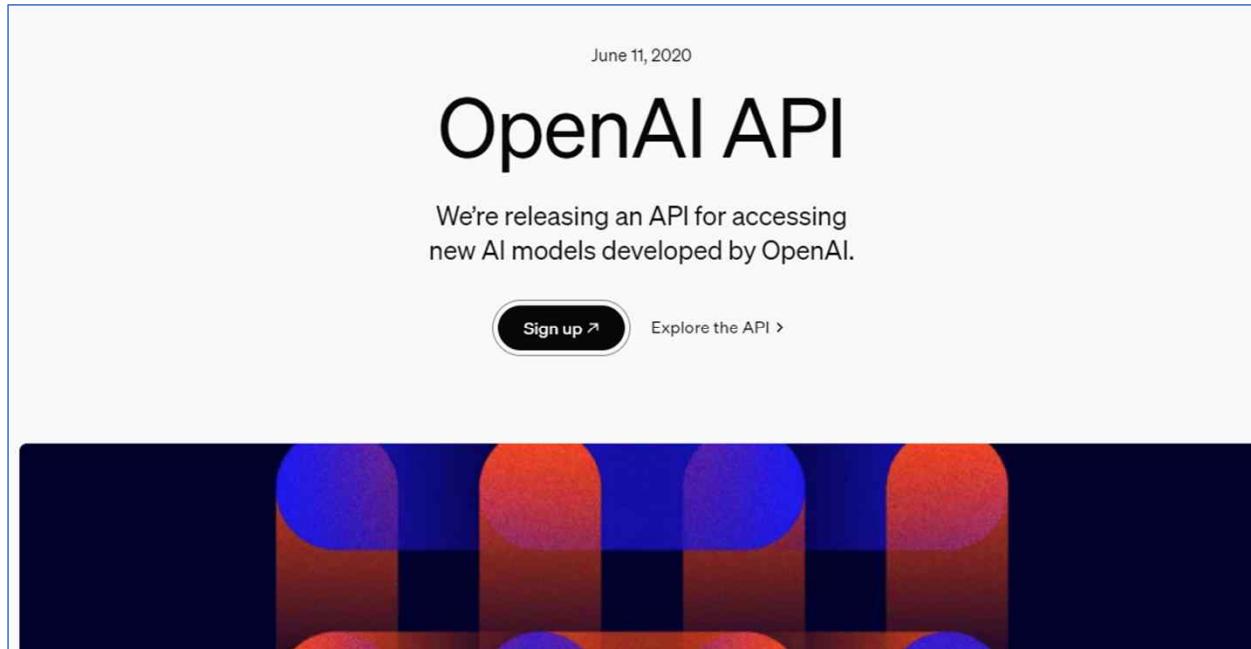
- API란 응용 프로그램 간의 상호 작용을 위한 인터페이스
- 복잡한 모델을 직접 구축하지 않고도 LLM의 기능 활용 가능
- 최신 모델의 기능을 손쉽게 통합 가능
- 주요 LLM API 제공업체 - OpenAI, Anthropic, Google DeepMind 등
- API 활용 시 고려사항
 - 요금 구조
 - 요청 제한
 - 보안 및 개인정보 보호

LangChain

- LLM을 활용한 애플리케이션 개발을 지원하는 프레임워크
- 자연어 처리, 데이터 연결, 메모리 관리, API 통합 등 다양한 기능 제공
- LangChain의 장점
 - 복잡한 LLM 기반 애플리케이션을 모듈화하여 개발
 - 다양한 외부 데이터 소스 및 API와의 통합 인터페이스 지원
 - 오픈소스로 활발한 커뮤니티와 지속적인 업데이트
- 단점
 - 복잡성 및 과도한 추상화로 개발 효율성을 저하
 - 추가적인 연산 부담으로 성능 저하.
 - 학습 곡선이 높아 초보자나 작은 프로젝트를 진행하는 개발자들에게 부담
 - 잦은 버전 변경으로 업그레이드 부담

Open-AI 계정 등록

<https://openai.com/index/openai-api/>



The screenshot shows the official landing page for the OpenAI API. At the top right, the date "June 11, 2020" is displayed. Below it, the title "OpenAI API" is prominently shown in a large, bold, black font. Underneath the title, a subtitle reads: "We're releasing an API for accessing new AI models developed by OpenAI." To the left of the subtitle is a "Sign up" button with a small arrow icon, and to the right is a link "Explore the API >". A decorative graphic of abstract, colorful shapes (blue, red, orange) is visible at the bottom of the page.

계정 만들기

이메일 주소* _____

계속

이미 계정이 있으신가요? [로그인](#)

_____ 또는 _____

 Google로 계속하기

 Microsoft 계정으로 계속하기

 Apple로 계속하기

Billing 정보 등록 (settings에서 등록)

The screenshot shows the OpenAI developer platform interface. At the top, there's a navigation bar with a user icon, 'Personal' dropdown, 'Default project' dropdown, 'Playground', 'Dashboard', 'Docs', 'API reference', and a gear icon with a green checkmark. A blue arrow points from the right towards the gear icon. On the left, a sidebar titled 'GET STARTED' contains links for 'Overview' (which is highlighted in light purple), 'Quickstart', 'Models', 'Pricing', 'Changelog', and 'Terms and policies'. Below that, under 'CAPABILITIES', are links for 'Text generation', 'Vision', 'Image generation', 'Audio generation', and 'Text to speech'. The main content area features the title 'OpenAI developer platform' and a 'Developer quickstart' section with a brief description and a '5 min' estimate. To the right of this is a code snippet for Python:

```
python ◊
1  from openai import OpenAI
2  client = OpenAI()
3  completion = client.chat.completions.create(
4      model="gpt-4o",
5      store=True,
6      messages=[
7          {"role": "user", "content": "write a haiku about ai"}
8      ]
9  )
```

At the bottom of the main content area, there are two buttons: 'Meet the models' and 'Pricing ↗'.

Billing 정보 등록 (Settings → Billing)

The screenshot shows the 'Billing' section of a user's settings. On the left sidebar, under 'SETTINGS', 'Billing' is selected. The main content area is titled 'Billing' and includes tabs for 'Overview', 'Payment methods', 'Billing history', and 'Preferences'. The 'Overview' tab is active. It displays a balance of '\$10.50' and a plan labeled 'Pay as you go' with a blue arrow pointing to the Korean translation '사용한 만큼 지불하는 방식 (Pay as you go)'. Below this, there's a note about a credit balance of '\$10.50' and a button to 'Add to credit balance'. A 'Cancel plan' button is also visible. A callout box highlights the 'Auto recharge is on' setting, which is checked, with a note that it will be triggered when the balance reaches \$5.00 to bring it up to \$15.00. A 'Modify' button is next to this note. At the bottom, there are links for 'Payment methods' (with an 'Add or change payment method' option) and 'Billing history' (with a 'View past and current invoices' link). The top navigation bar shows 'Personal' and 'Default project', and the right side has links for 'Playground', 'Dashboard', 'Docs', 'API reference', and a user profile icon.

Personal / Default project

Playground Dashboard Docs API reference

SETTINGS

Your profile

ORGANIZATION

General

API keys

Admin keys

Members

Projects

Billing

Overview Payment methods Billing history Preferences

Pay as you go → 사용한 만큼 지불하는 방식 (Pay as you go)

Credit balance ⓘ

\$10.50

Add to credit balance Cancel plan

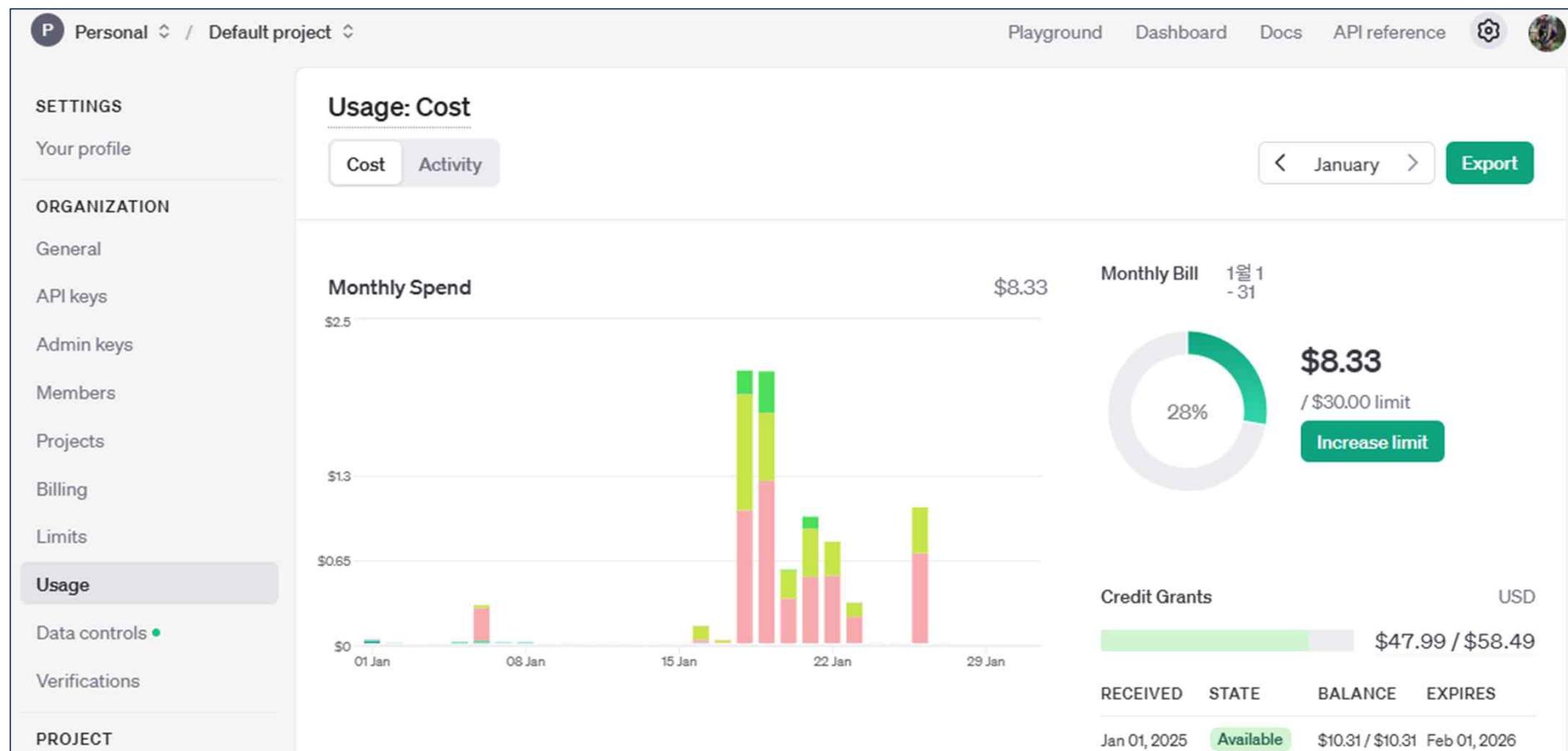
• 자동 충전 설정: 켜져 있으며, 크레딧 잔액이 \$5.00에 도달하면 결제 방식이 사용되어 잔액이 \$15.00로 충전됩니다.

Auto recharge is on When your credit balance reaches \$5.00, your payment method will be charged to bring the balance up to \$15.00. Modify

Payment methods Add or change payment method

Billing history View past and current invoices

사용량 조회 (Usage)



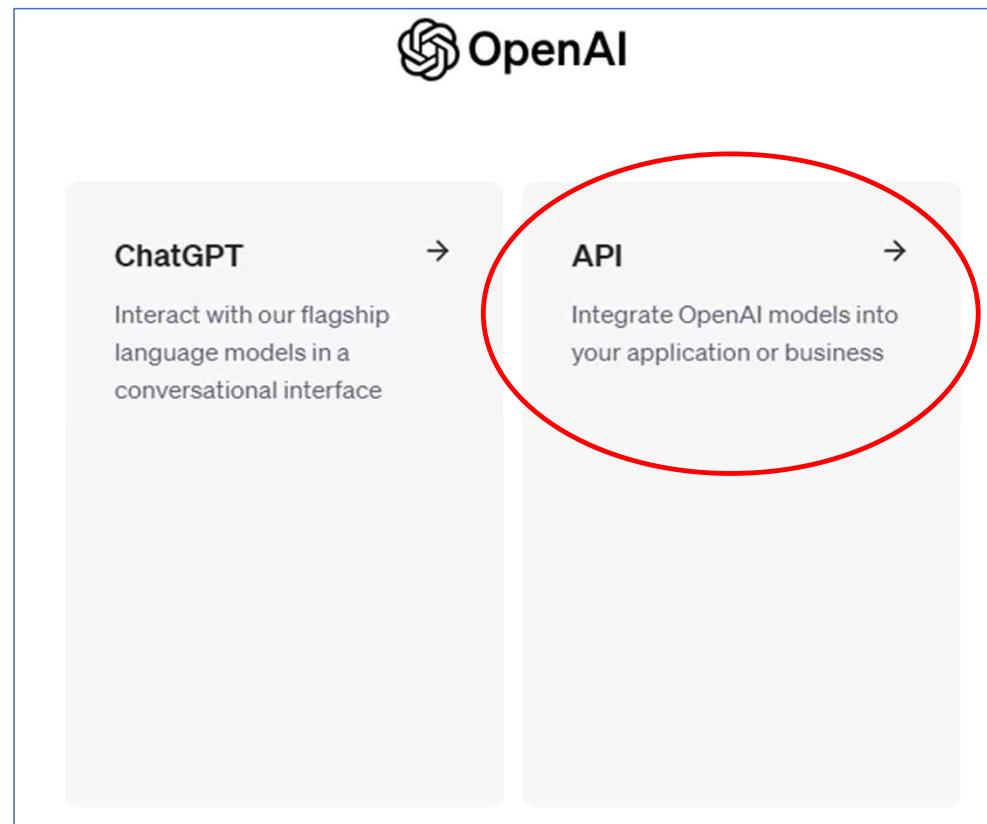
API 가격(<https://platform.openai.com/docs/pricing>)

The screenshot shows the 'Pricing' section of the OpenAI Platform. On the left, there's a sidebar with 'GET STARTED' and 'CAPABILITIES' sections. The 'Pricing' option in the sidebar is highlighted. The main content area has a title 'Pricing' and a sub-section 'Latest models'. It displays a table of 'Text tokens' with columns for Model, Input, Cached input, and Output. The table lists several models with their respective prices per 1M tokens.

Model	Input	Cached input	Output
gpt-4o ↳ gpt-4o-2024-08-06	\$2.50	\$1.25	\$10.00
gpt-4o-audio-preview ↳ gpt-4o-audio-preview-2024-12-17	\$2.50	-	\$10.00
gpt-4o-realtime-preview ↳ gpt-4o-realtime-preview-2024-12-17	\$5.00	\$2.50	\$20.00
gpt-4o-mini ↳ gpt-4o-mini-2024-07-18	\$0.15	\$0.075	\$0.60
gpt-4o-mini-audio-preview ↳ gpt-4o-mini-audio-preview-2024-12-17	\$0.15	-	\$0.60
gpt-4o-mini-realtime-preview ↳ gpt-4o-mini-realtime-preview-2024-12-17	\$0.60	\$0.30	\$2.40

Open-AI API Program

<https://platform.openai.com/apps>



API key 생성 (Dashboard → API keys)

<https://platform.openai.com/api-keys>

The screenshot shows the OpenAI Platform dashboard for a 'Default project'. On the left sidebar, the 'API keys' option is selected. The main content area is titled 'API keys' and contains instructions for managing API keys. A green button at the top right says '+ Create new secret key'. A large blue arrow points from the bottom of the page towards this button. Below the button, there's a table with one row showing a test key:

NAME	SECRET KEY	LAST USED	CREATED BY	PERMISSIONS
My Test Key	sk-...7XGD	Never	YoungJea Oh	All

A callout box highlights the text: '이 프로젝트의 소유자로서 귀하는 이 프로젝트의 모든 API 키를 보고 관리할 수 있습니다.' (As the owner of this project, you can view and manage all API keys in this project.)

Below the table, another callout box contains the following text:

API 키를 다른 사람과 공유하거나 브라우저 또는 기타 클라이언트 측 코드에 노출하지 마십시오. 귀하의 계정 보안을 보호하기 위해 OpenAI는 공개적으로 유출된 모든 API 키를 자동으로 비활성화할 수도 있습니다.

사용량 페이지에서 API 키별 사용량을 확인하세요.

Key는 한번만 보여주므로 메모장 등에 복사한 후 .env 파일로 저장

Create new secret key

Owned by
 You Service account

이 API 키는 사용자와 연결되어 있으며 선택한 프로젝트에 대해 요청할 수 있습니다. 조직이나 프로젝트에서 제거되면 이 키가 비활성화됩니다.

Name Optional
My Test Key

Project
Default project

Permissions
[All](#) [Restricted](#) [Read Only](#)

[Cancel](#) [Create secret key](#)

Save your key

Please save this secret key somewhere safe and accessible. For security reasons, **you won't be able to view it again** through your OpenAI account. If you lose this secret key, you'll need to generate a new one.

`3RuDSd7TQbDNViu39T3qT3B1bkFJEoTJIwcT8PhptPA:` [Copy](#)

Permissions
Read and write API resources

[Done](#)

이 프로젝트의 소유자로서 귀하는 이 프로젝트의 모든 API 키를 보고 관리할 수 있습니다.								
API 키를 다른 사람과 공유하거나 브라우저 또는 기타 클라이언트 쪽 코드에 노출하지 마십시오. 귀하의 계정 보안을 보호하기 위해 OpenAI는 공개적으로 유출된 모든 API 키를 자동으로 비활성화할 수도 있습니다.								
사용량 페이지 에서 API 키별 사용량을 확인하세요.								
<table><thead><tr><th>이름</th><th>비밀키</th><th>작성자:</th><th>권한</th></tr></thead><tbody><tr><td>내 테스트 키</td><td>sk-...7XGD</td><td>오영재</td><td>모두</td></tr></tbody></table>	이름	비밀키	작성자:	권한	내 테스트 키	sk-...7XGD	오영재	모두
이름	비밀키	작성자:	권한					
내 테스트 키	sk-...7XGD	오영재	모두					

생성된 API Key 관리

- 타인 노출 안되도록 조심
- 노출되면 반드시 삭제 후 재 생성

API keys

+ Create new secret key

As an owner of this project, you can view and manage all API keys in this project.

Do not share your API key with others or expose it in the browser or other client-side code. To protect your account's security, OpenAI may automatically disable any API key that has leaked publicly.

View usage per API key on the [Usage page](#).

NAME	SECRET KEY	LAST USED ⓘ	CREATED BY	PERMISSIONS
My Test Key	sk-...7XGD	Never	YoungJea Oh	All

단일 프로젝트에 대해 API key 설정 방법

- .env – API 키가 포함된 로컬 파일 생성

```
# .env 파일  
OPENAI_API_KEY=sk-vvDtll*****XiiIEpdjLhBJaH0f
```

- .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.environ에 OPENAI_API_KEY 추가
```

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

OpenAI API 주요 Component

- 텍스트 생성 모델
 - 자연어와 형식 언어를 이해하도록 훈련된 모델 (예, GPT-4, GPT-4o, o1, etc)
 - 프롬프트 - 입력 텍스트
 - 콘텐츠 생성, 코드 작성, 요약, 대화, 창의적 글쓰기 등
- 어시스턴트 (Assistants)
 - 대화형 AI 개발 지원 → 복잡한 대화 시나리오를 구현 가능
 - 컨텍스트 유지 → 대화의 맥락을 유지하며 자연스럽게 여러 턴의 대화를 이어감.
 - 지시사항 및 사용자 정의 → 특정한 방식으로 응답하도록 모델을 조정 가능.
 - 코드 접근 → 코드 실행, 파일에서 정보 검색 등을 수행할 수 있는 도구에 접근

- 임베딩 (Embedding)
 - 텍스트 임베딩 모델을 이용하여 임베딩 벡터를 출력으로 생성
 - 데이터 조각(텍스트)의 벡터 표현으로 의미를 보존.
 - 유사한 데이터는 더 가까운 임베딩을 가짐.
 - 검색, 클러스터링, 추천, 이상 탐지, 분류 등에 유용.
- 토큰 (Token)
 - 텍스트를 처리하는 단위로, 문자 시퀀스를 나타냄.
 - 예) "tokenization"은 "token" + "ization"으로 분해됨.
 1 토큰은 영어 텍스트의 경우 약 4자 또는 0.75단어.
 - 토큰의 길이는 모델의 최대 컨텍스트 길이에 따라 제한됩니다.

Models

모델	설명
GPT-4o	<ul style="list-style-type: none">- 텍스트와 이미지 입력을 모두 받아들이며, 텍스트 출력(구조화된 출력 포함)을 생성할 수 있습니다.- chatgpt-4o-latest 모델 ID는 ChatGPT에서 사용되는 GPT-4o 버전을 지속적으로 가리킵니다. 이 모델은 ChatGPT의 GPT-4o에 중요한 변경 사항이 있을 때마다 자주 업데이트됩니다.
GPT-4o mini	<ul style="list-style-type: none">- 빠르고 경제적인 소형 모델로, 텍스트와 이미지 입력을 모두 받아들이며, 텍스트 출력(구조화된 출력 포함)을 생성할 수 있습니다.
o1 and o1-mini	<ul style="list-style-type: none">- o1 시리즈 모델은 복잡한 추론을 수행하도록 강화 학습을 통해 훈련되었습니다.- o1 : 다양한 분야에서 난해한 문제를 해결하도록 설계된 추론 모델- o1-mini : 빠르고 경제적인 추론 모델
DALL-E3	주어진 프롬프트를 바탕으로 특정 크기의 새로운 이미지를 생성할 수 있습니다.
TTS	텍스트를 자연스러운 음성으로 변환할 수 있는 모델
Whisper	오디오를 텍스트로 변환할 수 있는 모델
Embeddings	텍스트를 embedding vector로 변환할 수 있는 모델

Chat Completions API

- OpenAI의 Chatting API를 사용하여 다음과 같은 응용 프로그램을 구축할 수 있다.
 - 문서 초안 작성
 - 컴퓨터 코드 작성
 - 지식 기반에 대한 질문에 답하기
 - 텍스트 분석
 - 소프트웨어에 자연어 인터페이스 제공
 - 다양한 과목에 대한 튜터링
 - 언어 번역
 - 게임용 캐릭터 시뮬레이션

OpenAI API role 의 종류

역할(role)	설명	예제
user	모델에게 특정 출력을 요청하는 메시지. ChatGPT를 사용할 때 유저가 입력하는 일 반적인 메시지와 동일함.	"프로그래밍에 대한 시를 한편 작성해줘."
developer	사용자 메시지보다 우선적으로 적용되는 모델 지침. 이전에는 "system prompt"라고 불렸음.	"당신은 유용한 프로그래밍 도우미입니다."
assistant	모델이 생성한 응답 메시지. 이전 대화에서 생성된 메시지를 포함할 수도 있음.	"똑똑! 누구세요? 나는 OpenAI야! OpenAI가 누구야?"

이러한 역할은 대화형 세션에서 메시지의 발신자와 의도를 구별하는 데 도움을 준다. API는 이 정보를 사용하여 적절한 방식으로 반응하거나 응답.

실습: 320. Text Generation

- Chat Completion API 사용
- 메시지 역할 설명 - user, developer, assistant
- 구조화된 출력 (Structured Outputs)
- 메시지에서 사용된 토큰 수 계산 및 관리
- SEED 매개변수를 사용하여 재현 가능한 출력 생성
- 모델을 사용하여 간단한 수학 문제 해결

감사합니다.