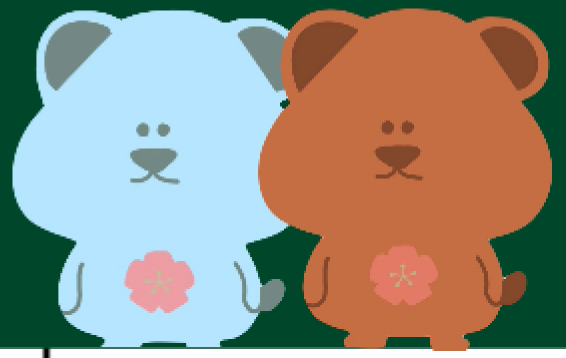
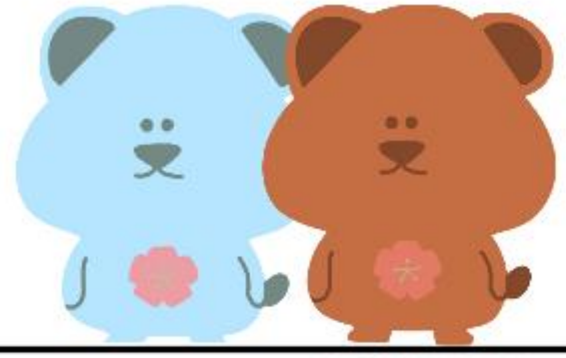


Week9: Coronavirus tweets NLP

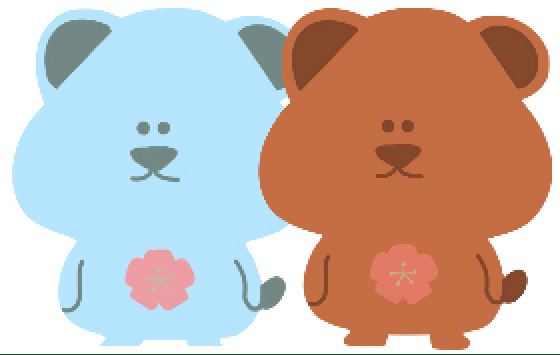
발표자: 이지호, 김희숙

EWHA EURON

목차



대회 소개



#01 대회 소개 coronavirus tweets NLP

대회 개요

- 트위터에서 수집한 코로나 바이러스에 관한 텍스트에 감성 분석 태그를 분류하는 텍스트 분류문제.

데이터 셋

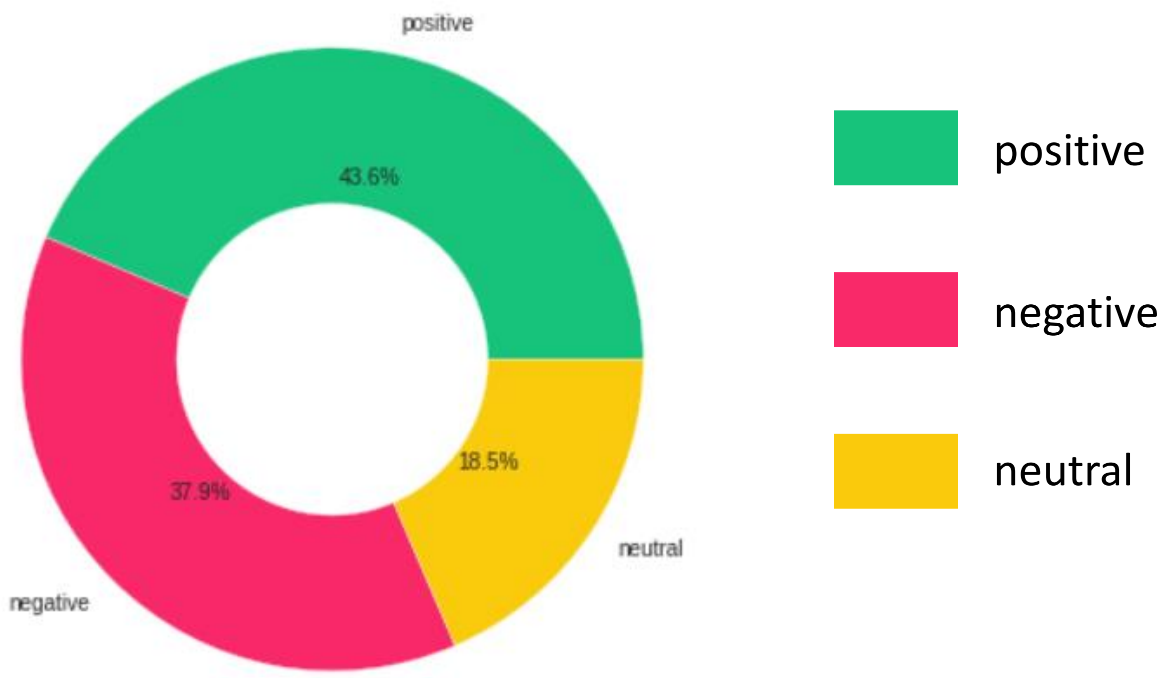
- Columns: Username, ScreenName, Location, TweetAt, Original Tweet, Sentiment(classification label)

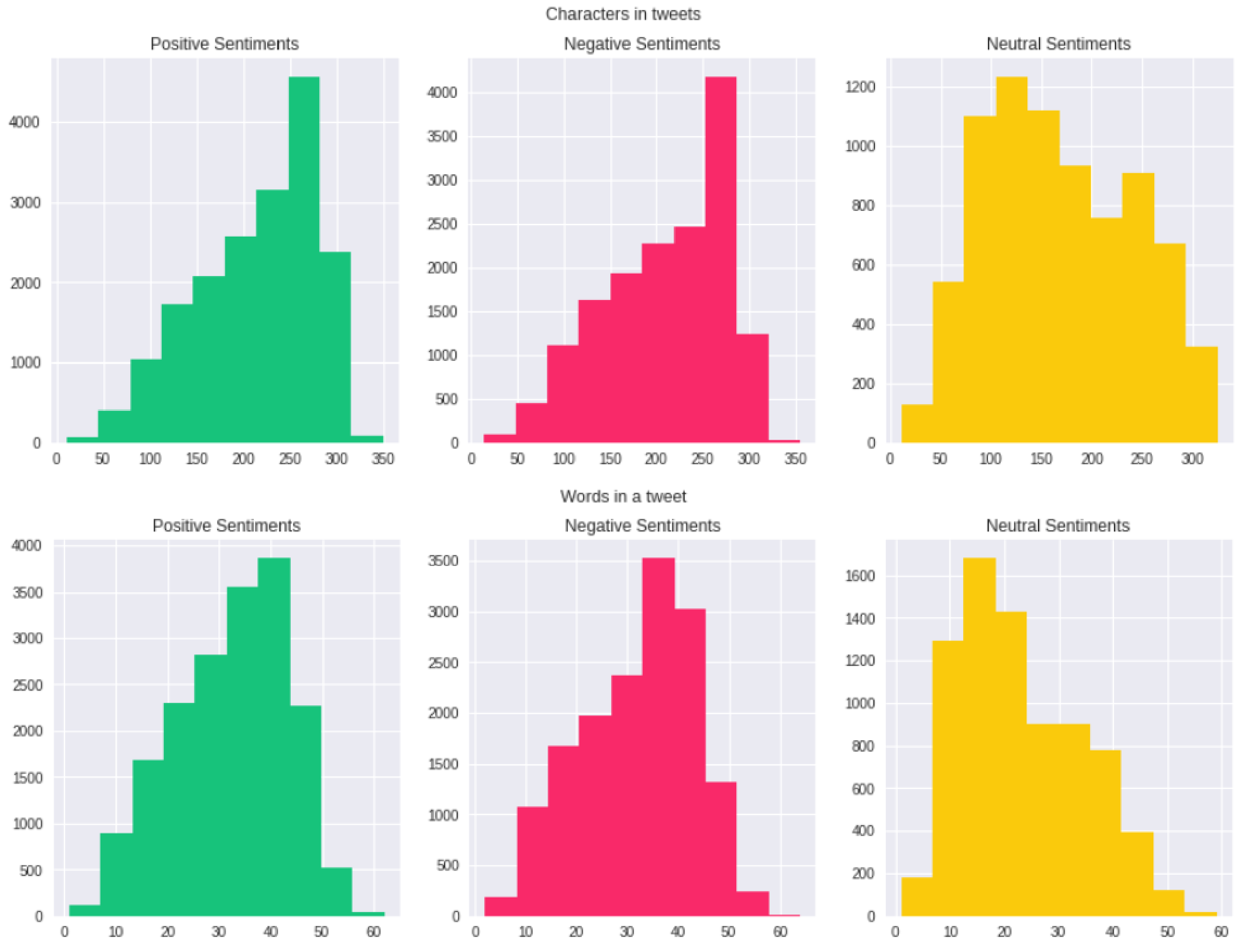
# UserName	# ScreenNa...	A Location	A TweetAt	A OriginalTw...	A Sentiment
3843	48795		16-03-2020	This is the line outside @Target in as customers wait for the store to open this morning	Neutral
3844	48796	Midrand	16-03-2020	South Africans stock up on food, basic goods as coronavirus panic hits https://t.co/6nGNFJmy89 ...	Negative
3845	48797	Drogheda	16-03-2020	Please Share Know someone who s 65 Living on their own struggling to get 2 their local supermarket...	Extremely Positive

#01 대회 소개 coronavirus tweets NLP

Data EDA

```
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   UserName       41157 non-null  int64
 1   ScreenName     41157 non-null  int64
 2   Location       32567 non-null  object
 3   TweetAt        41157 non-null  object
 4   OriginalTweet  41157 non-null  object
 5   Sentiment      41157 non-null  object
dtypes: int64(2), object(4)
```

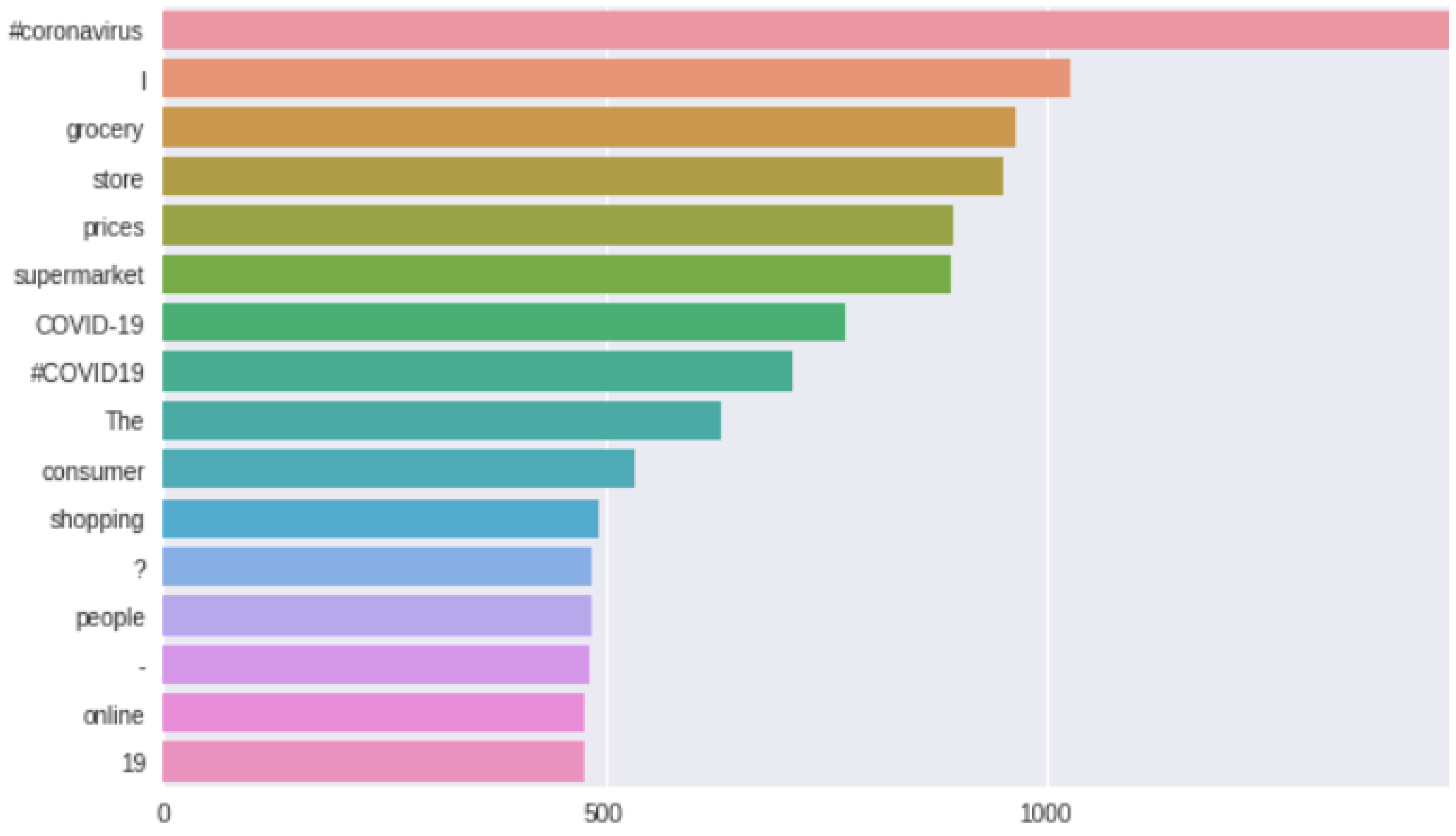




Words, len of sentence.

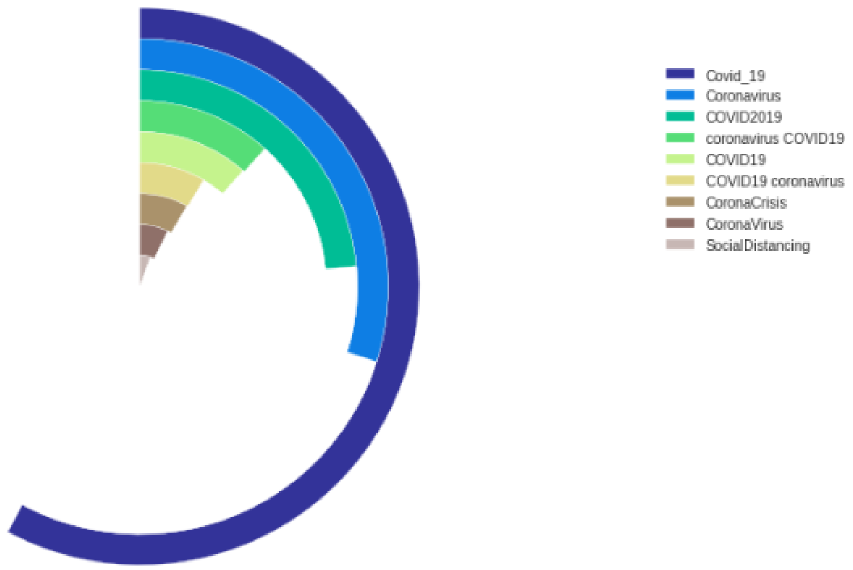
#01 대회 소개 coronavirus tweets NLP

Data EDA




```
np.array(stop)
```

```
array({'such', "hadn't", 'needn', 'same', 'where', 'were',
'over', 'now', 'if', "you'll", 'very', 'he', 'its', 'have
n', "mightn't", 's', 'did', 'up', 'was', 'in', 'themselve
s', 'wouldn', 'how', 'which', "couldn't", 'll', 'd', 'she',
'we', 'what', 'i', 'on', 'didn', 'there', 'who', 'don', 'o
f', 'isn', 'an', 'couldn', 'had', 'before', "shan't", 'ow
n', 'here', 'off', 'their', 've', 'until', 'both', 'o', 'hi
s', 'being', 'than', "didn't", 'a', 'not', 'himself', 'sha
n', "should've", 'this', 'm', "wasn't", 'are', 'above', 'ha
ve', 'they', 'no', 'by', 'other', 'weren', "hasn't", 'it',
'for', 't', 'hers', 'down', "aren't", 'is', 'itself', "yo
u're", 'been', 'that', 'him', 'out', 'the', 'yourselves',
'then', "mustn't", 'at', "you've", 'as', 'our', 'during',
'will', 'few', 'once', 'most', "you'd", 'ain', 'into', 'you
rself', 'only', 'shouldn', 'you', 'to', "doesn't", 'wasn',
```



- ➢ 자주 등장하는 단어들: #coronavirus, I, grocery, store, prices, supermarket, COVID-19, #COVID19, The, consumer, shopping, ?, people, -, online, 19
- ➢ Stopwords
- ➢ Lower caching needed

#01 대회 소개 coronavirus tweets NLP

Typical Preprocessing

1. Lower capital alphabets

```
# Lower casing
def lower(text):
    low_text= text.lower()
    return low_text
df['text_new']=df['text'].apply(lambda x:lower(x))
```

2. Remove or mapping asterisk

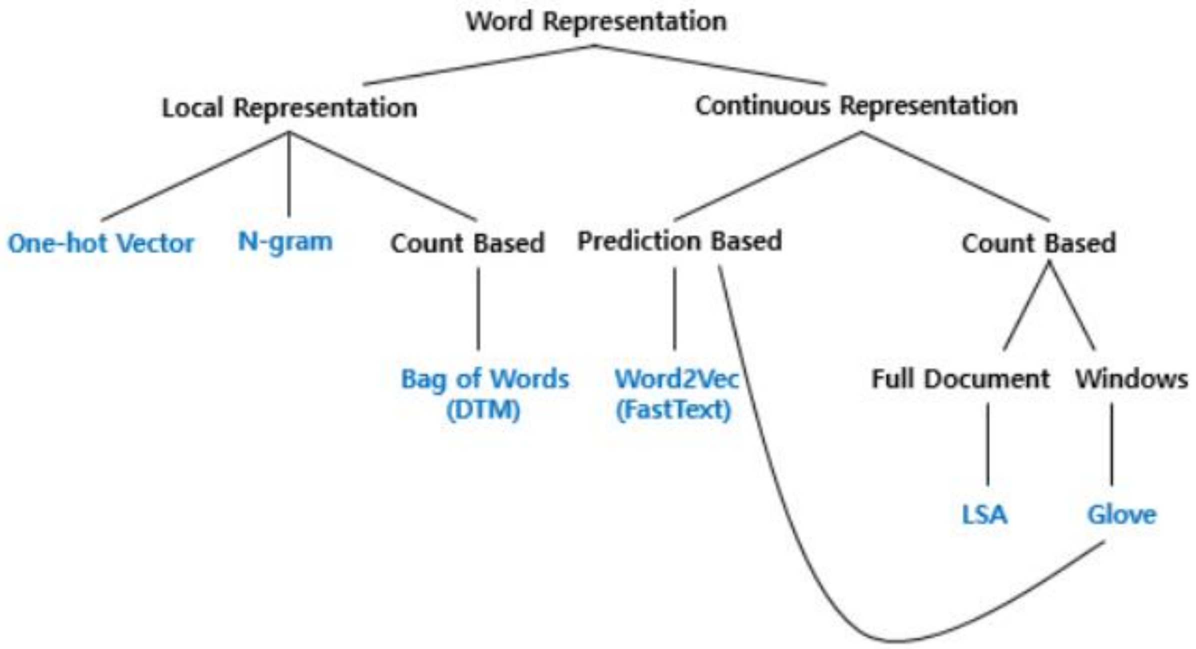
```
# Number removal
def remove_num(text):
    remove= re.sub(r'\d+', '', text)
    return remove
df['text']=df['text_new'].apply(lambda x:remove_num(x))
```

3. Remove stopwords & punctuations

```
#Remove stopwords & Punctuations
from nltk.corpus import stopwords
", ".join(stopwords.words('english'))
STOPWORDS = set(stopwords.words('english'))

def punct_remove(text):
    punct = re.sub(r"[^\w\s\d]","", text)
    return punct
df['text_new']=df['text'].apply(lambda x:punct_remove(x))



def remove_stopwords(text):
    """custom function to remove the stopwords"""
    return " ".join([word for word in str(text).split() if wor
d not in STOPWORDS])
df['text']=df['text_new'].apply(lambda x:remove_stopwords(x))
```

#02 Language model: vectorization

Preprocessing: vectorization



- **Integer / one-hot encoding**

The cat sat on the mat

The: [0 1 0 0 0 0 0]

cat: [0 0 1 0 0 0 0]

sat: [0 0 0 1 0 0 0]

on: [0 0 0 0 1 0 0]

the: [0 0 0 0 0 1 0]

mat: [0 0 0 0 0 0 1]

```
One Hot Representation for sentence "the cat sat on the mat" :


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

#02 Language model: vectorization

Preprocessing: vectorization

- BOW(bag of words)

```
Vocabulary mapping for given sample corpus :
 {'the': 4, 'cat': 0, 'sat': 3, 'in': 2, 'hat': 1, 'with': 5}

Bag of word Representation of sentence 'the cat cat sat in the hat'
[[2 1 1 1 2 0]]
```

➢ 각 단어가 인덱스에 매핑 되어있고 해당 단어의 등장 횟수를 기록함

- TF-IDF(bag of words)

TF-IDF(Term Frequency - Inverse Document Frequency)는 정보 검색과 텍스트 마이닝에서 이용하는 가중치로, 여러 문서로 이루어진 문서군이 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. 문서의 핵심어를 추출하거나, 검색 엔진에서 검색 결과의 순위를 결정하거나, 문서들 사이의 비슷한 정도를 구하는 등의 용도로 사용할 수 있다.

TF(단어 빈도, term frequency)는 특정한 단어가 문서 내에 얼마나 자주 등장하는지를 나타내는 값으로, 이 값이 높을수록 문서에서 중요하다고 생각할 수 있다. 하지만 단어 자체가 문서군 내에서 자주 사용 되는 경우, 이것은 그 단어가 흔하게 등장한다는 것을 의미한다. 이것을 DF(문서 빈도, document frequency)라고 하며, 이 값의 역수를 IDF(역문서 빈도, inverse document frequency)라고 한다. TF-IDF는 TF와 IDF를 곱한 값이다.

➢ Df는 문서 내 빈도가 아니라 문서 간 빈도, IDF는 DF와 반비례 $idf(d,t) = log(\frac{n}{1+df(t)})$

#02 Language model: vectorization

- **TF-IDF**

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

$$idf(d,t) = log(\frac{n}{1+df(t)})$$

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

#02 Language model: word2vec

- **Word2vec(continuous vectorization)**
  - Window 주변 단어들을 보고 단어를 예측하는 방법
  - CBOW & skip-gram
  - Embedding table(matrix)의 인풋

중심 단어

주변 단어

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

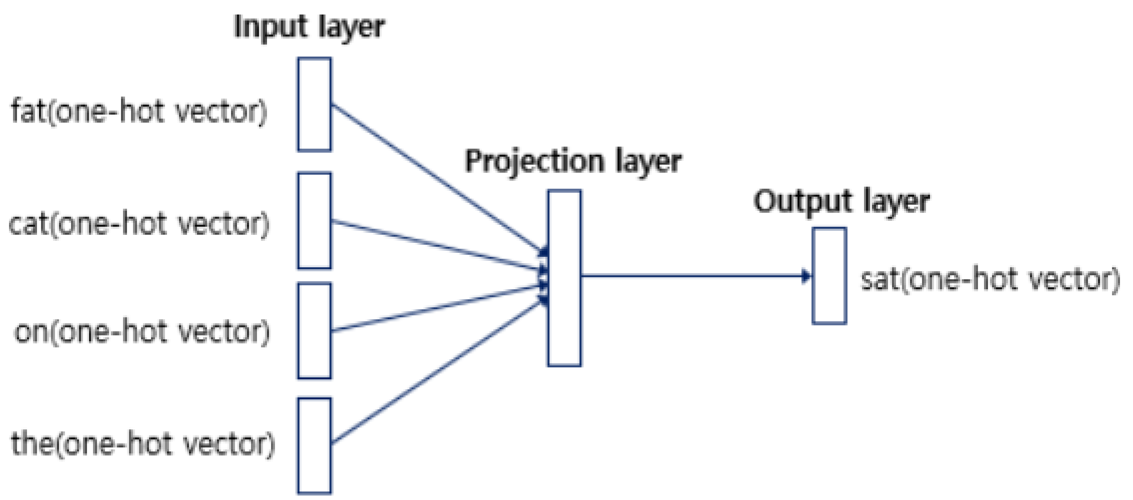
The fat cat sat on the mat

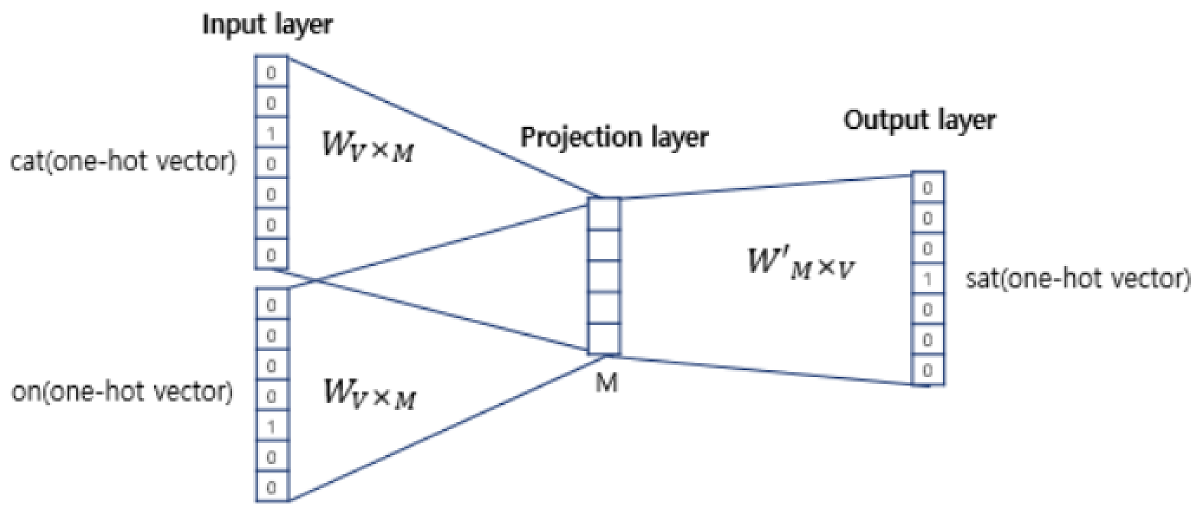
The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

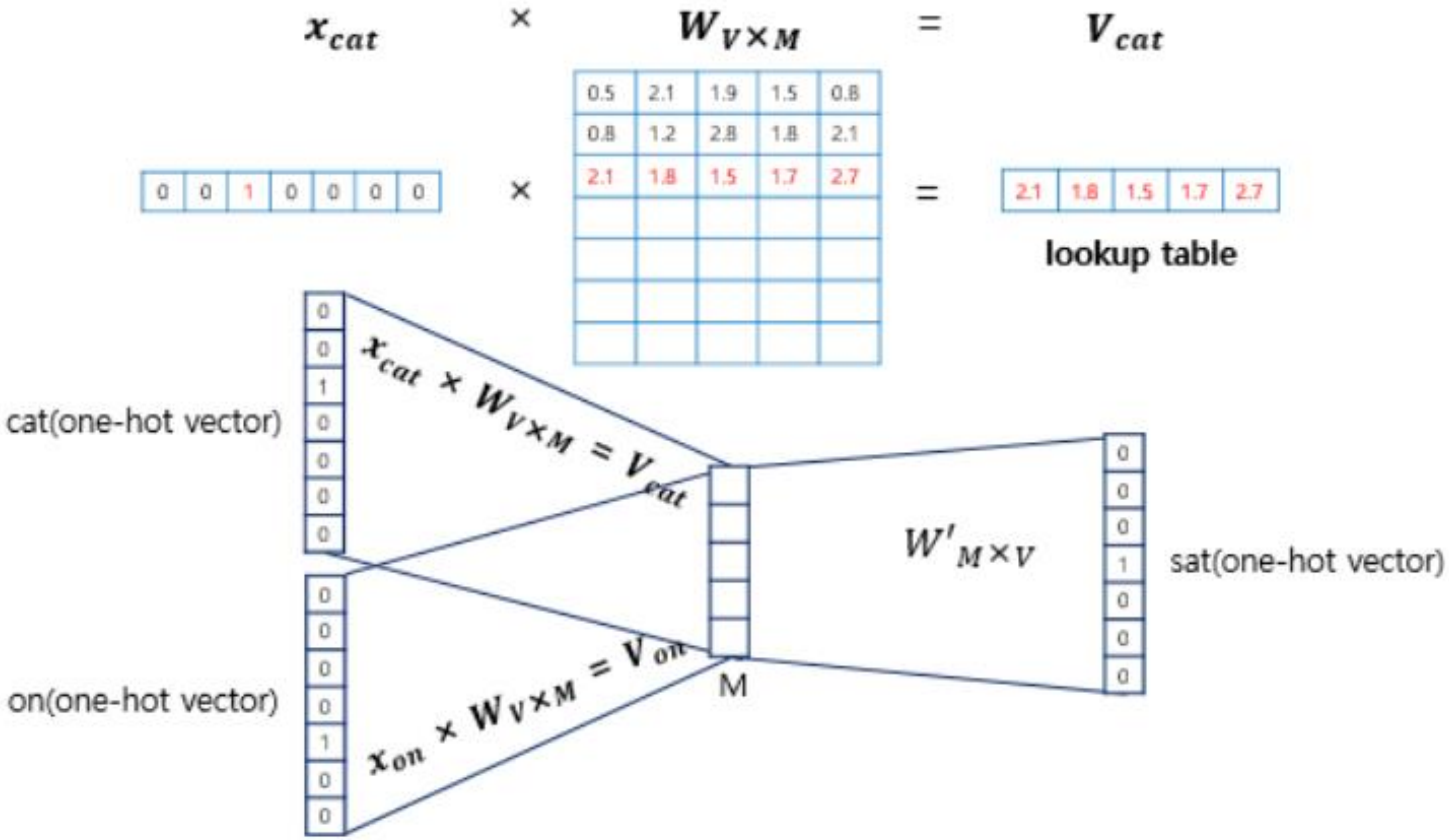
중심 단어	주변 단어
[1, 0, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0, 0]	[0, 1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0, 0]	[0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1, 0]





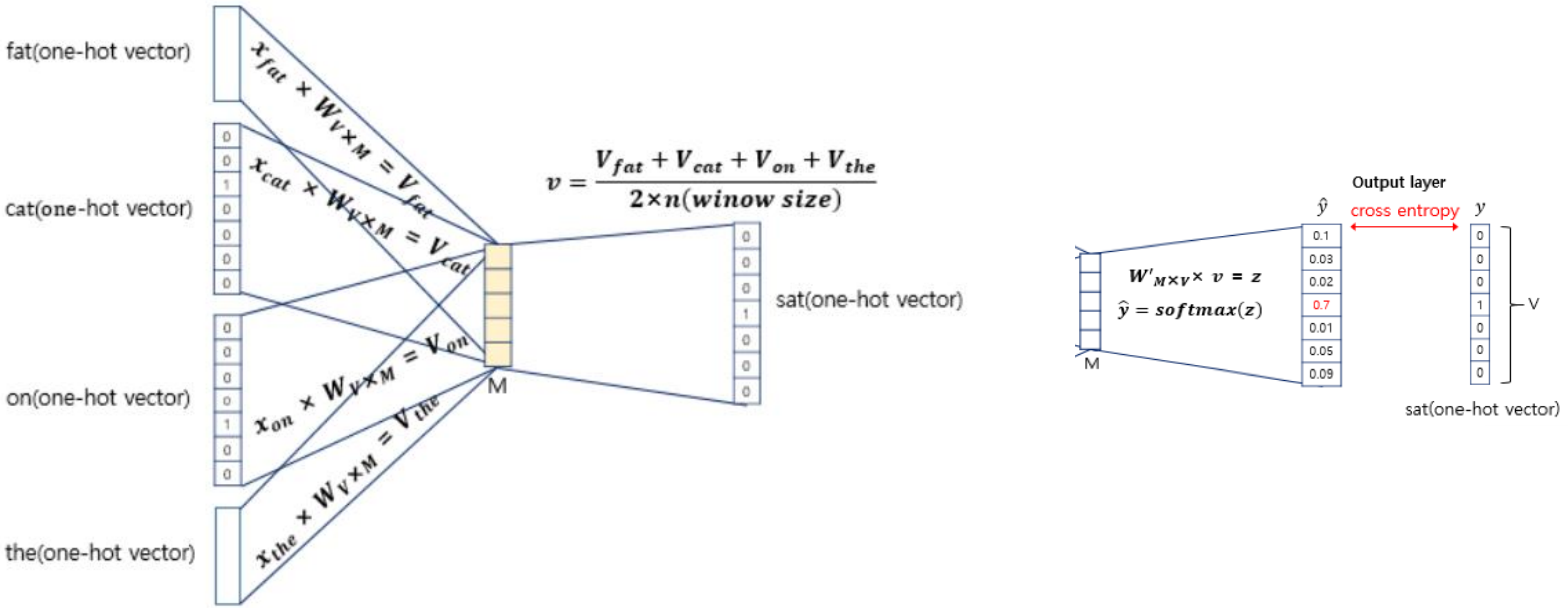
#02 Language model: word2vec

- **word2vec**



#02 Language model: word2vec

- **word2vec**



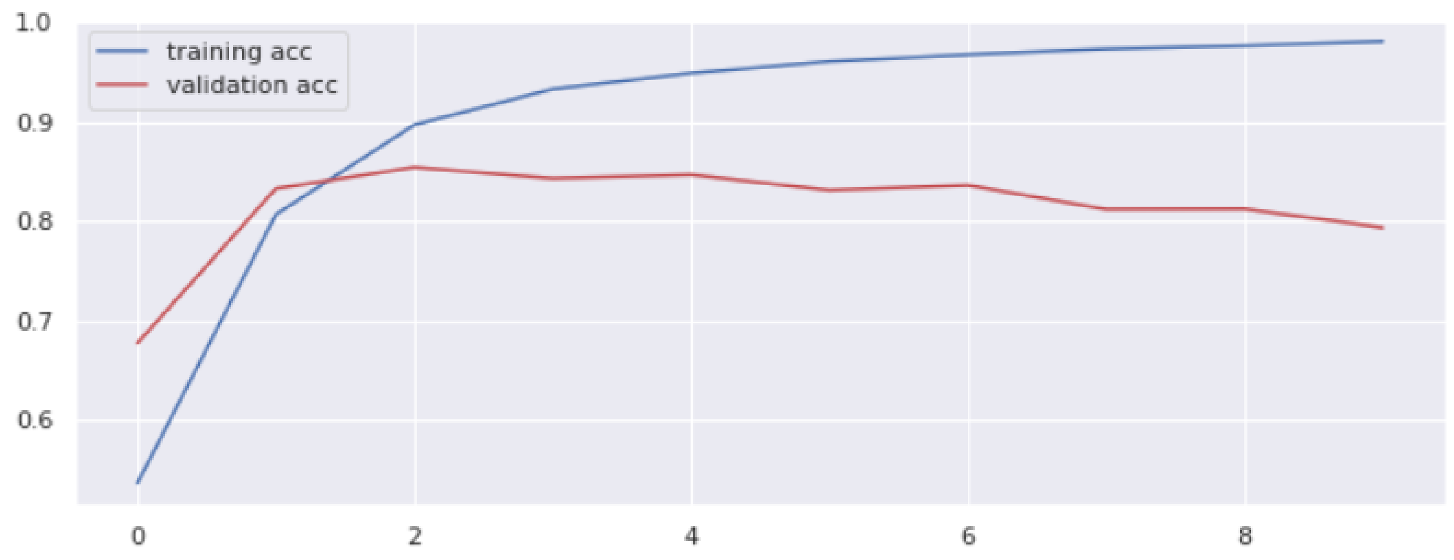
#02 Language model: word2vec

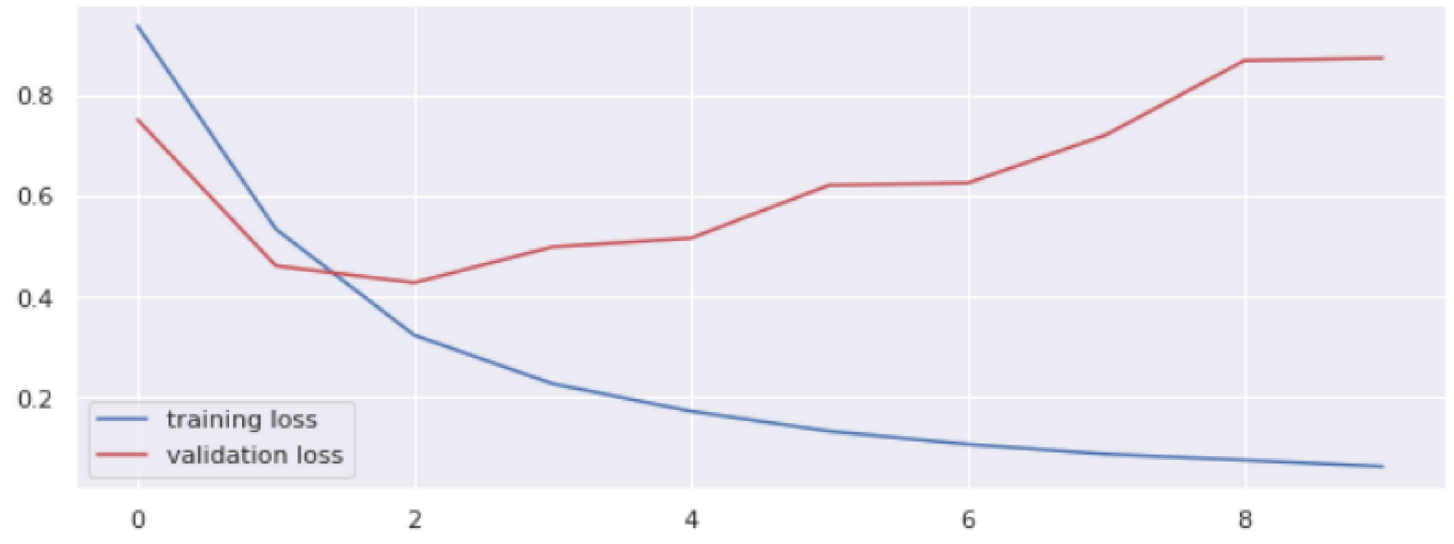
- **This is (Word) Embedding example**

➢ 워드 임베딩(Word Embedding)은 단어를 벡터로 표현하는 방법으로, 단어를 밀집 표현으로 변환합니다. 희소 표현, 밀집 표현, 그리고 워드 임베딩에 대한 개념을 학습합니다.
➢ 즉, 앞선 슬라이드에서 본 cbow, skip-gram등의 방식으로 word2vec 의 embedding table 가중치를 학습
➢ 최종적으로 얻은 embedding matrix에 각 단어를 변환 시켜 얻은 벡터 값
➢ 벡터들 간의 유사도를 정보로 사용
➢ module

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors

model = Word2Vec(sentences=result, size=100, window=5, min_count=5, workers=4, sg=0)
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_length, embedding_dim, inp
ut_length=max_len),
    tf.keras.layers.Bidirectional(tf.keras.layers.GRU(256, ret
urn_sequences=True)),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.Dense(3, activation='softmax')
])
# opt = tf.keras.optimizers.Adam(learning_rate=0.01)
model.compile(loss='categorical_crossentropy',optimizer="ada
m",metrics=['accuracy'])
```





#02 Language model

Statistical language model: N-grams

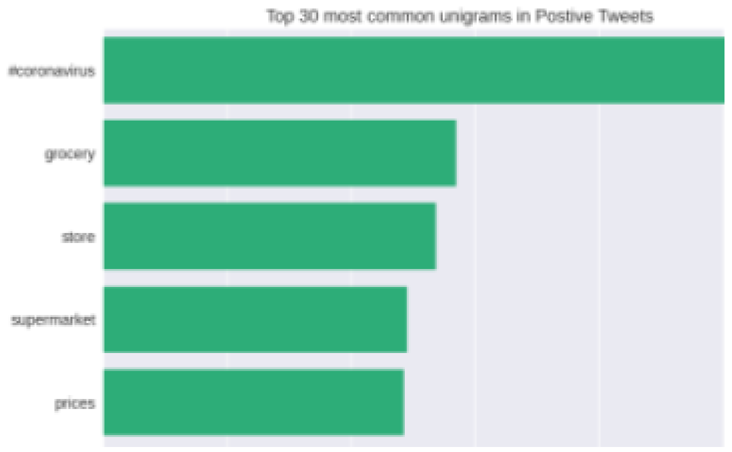
- 카운트 기반 단어 통계: N-grams

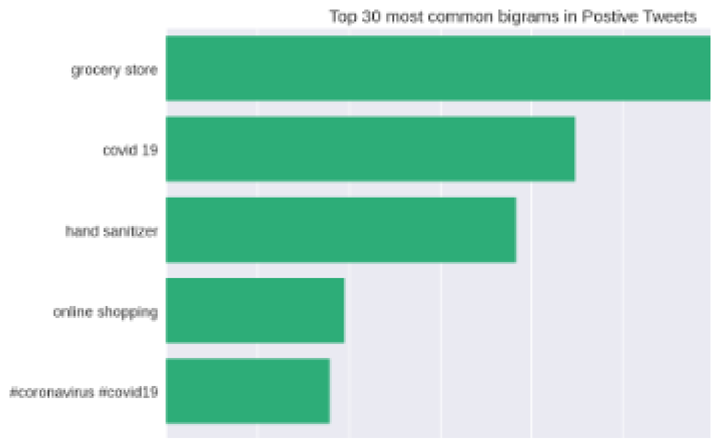
machine	fun	is	learning	and	not	boring	...
1	1	2	1	1	1	1	...

machine learning	learning is	is fun	fun and	and is	is not	not boring	...
1	1	2	1	1	1	1	...

➢ Count sparsity problem: uni-gram → bi-gram → tri-gram…

➢ 5 most common unigrams/bigrams in positive tweets





<uni-grams>
1. #coronavirus
2. Grocery
3. Store
4. Supermarket
5. prices

<bi-grams>
1. Grocery store
2. Covid 19
3. Hand sanitizer
4. Online shopping
5. # coronavirus #covid19

```python
# Define functions
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(' ') if token != '' if token not in STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [' '.join(ngram) for ngram in ngrams]
```

```python
# Unigrams
for tweet in train[positive]['text']:
    for word in generate_ngrams(tweet):
        positive_unigrams[word] += 1

for tweet in train[negative]['text']:
    for word in generate_ngrams(tweet):
        negative_unigrams[word] += 1

for tweet in train[neutral]['text']:
    for word in generate_ngrams(tweet):
        neutral_unigrams[word] += 1
```

#02 Language model: ML(non-neural network) classifier

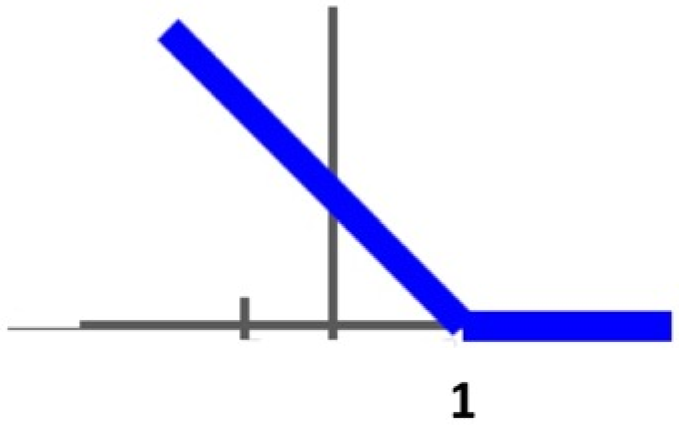
- ML Classifier - SVM
  - Hinge loss

- Binary hinge loss (=binary SVM loss)

$$L_i = \max(0, 1 - y_i \cdot s)$$

$$s = \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x_i} + b$$

$y_i = \pm 1$ for positive/negative samples



- Hinge loss (=multiclass SVM loss)
  - n: The number of class (> 2)

$$L_i = \sum_{j=1, j \neq y_i}^{n} \max\left(0, s_j - s_{y_i} + 1\right)$$

$$\boldsymbol{s} = \mathbf{W} \boldsymbol{x}_i + \boldsymbol{b}$$

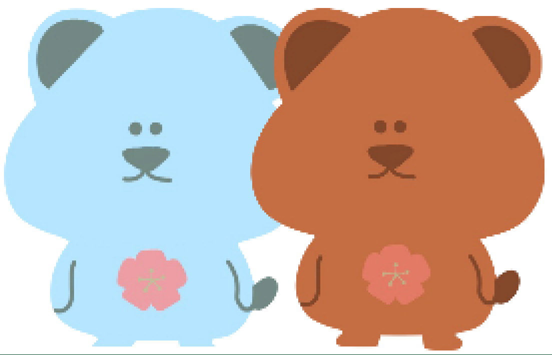
$$\mathbf{W} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_n^T \end{pmatrix} \qquad \boldsymbol{s} = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{pmatrix}$$

$\boldsymbol{x}_i$: input data (e.g. image)
y_i: class label (integer, $1 \leq y_i \leq n$)

#02 Language model: ML(non-neural network) classifier

- ML Classifier VS DL classfier
  - DL의 성능이 더 좋은 이유: data driven, feature training
  - ML: feature design, vectorization, calculate similarity
  - DL: model trains both feature + classifier
  - DL에서 sota NLP model 'BERT' → large model, embedding

BERT Model



EWHA EURON

#03 BERT embedding을 사용한 분석

Twitter Sentiment Analysis with BERT + RoBERTa 🐦

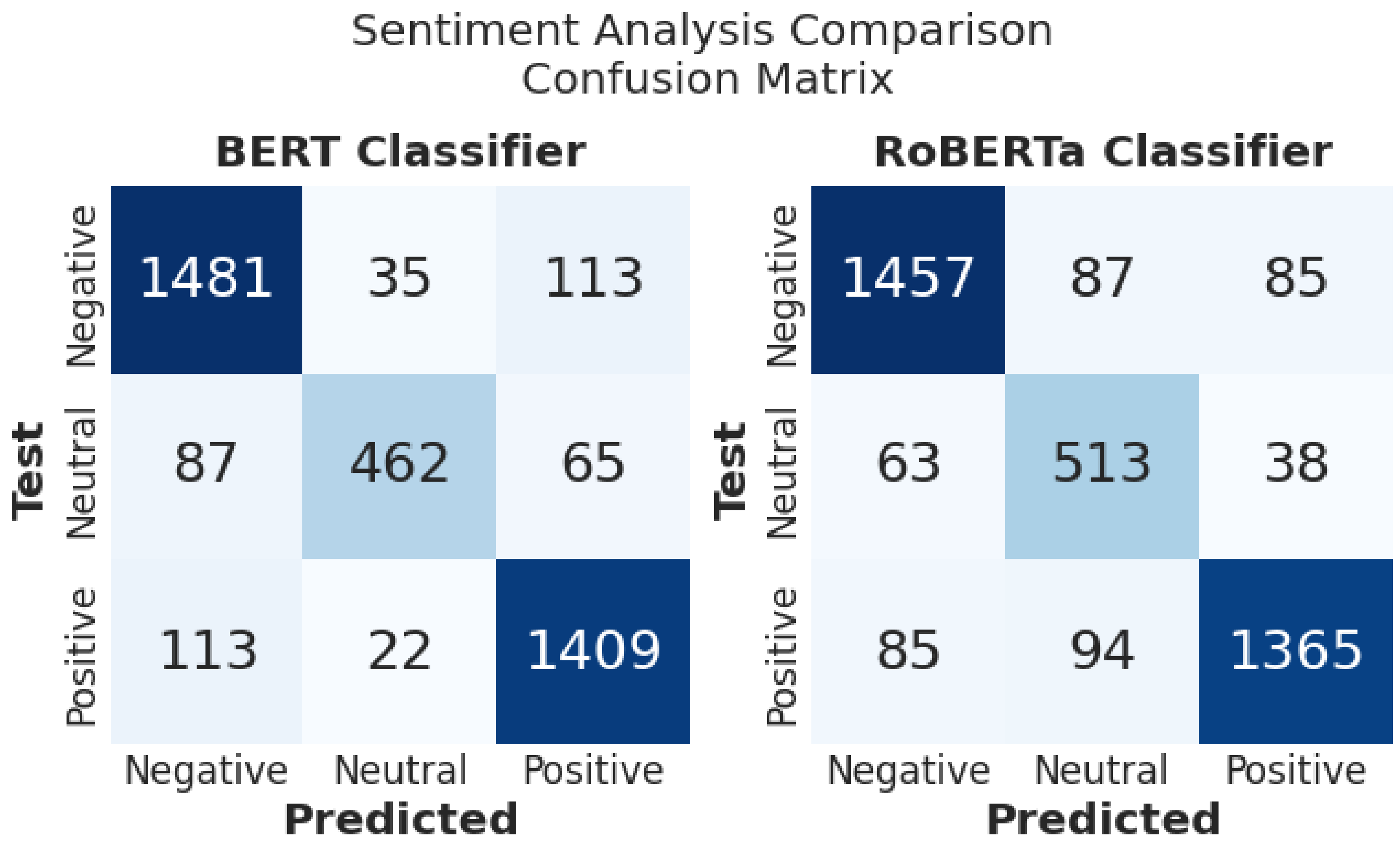
Twitter Sentiment Analysis with BERT + RoBERTa 🐦

Updated 4mo ago

40 comments · Coronavirus tweets NLP - Text Classification

▲ 65

Silver •••



- 데이터 전처리: 트윗에 포함된 링크, 해시태그, 구두점 제거
- 코로나 바이러스 트윗의 감정을 예측하기 위해 **BERT**와 **RoBERTa** 알고리즘 사용

https://www.kaggle.com/code/ludovicocuoghi/twitter-sentiment-analysis-with-bert-roberta

#03 BERT

Bidirectional Encoder Representations from Transformers

: designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers

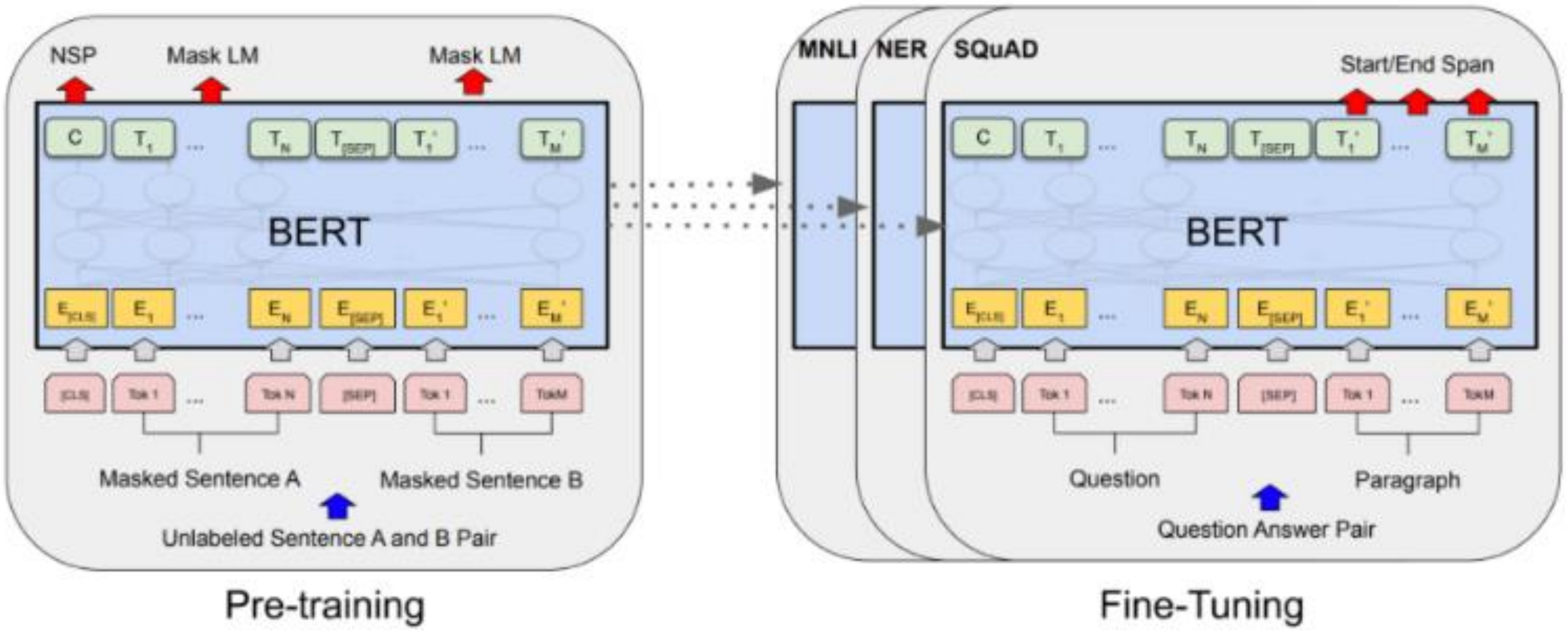


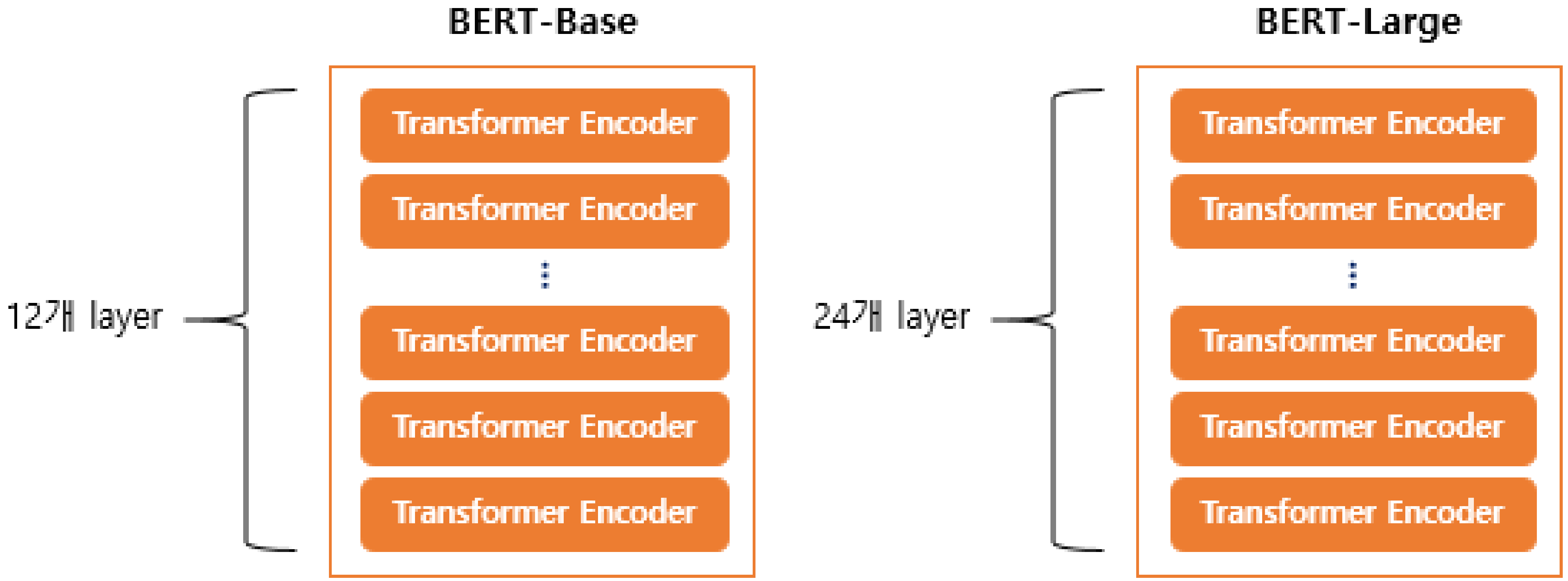
Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

#03 BERT Architecture

BERT의 크기

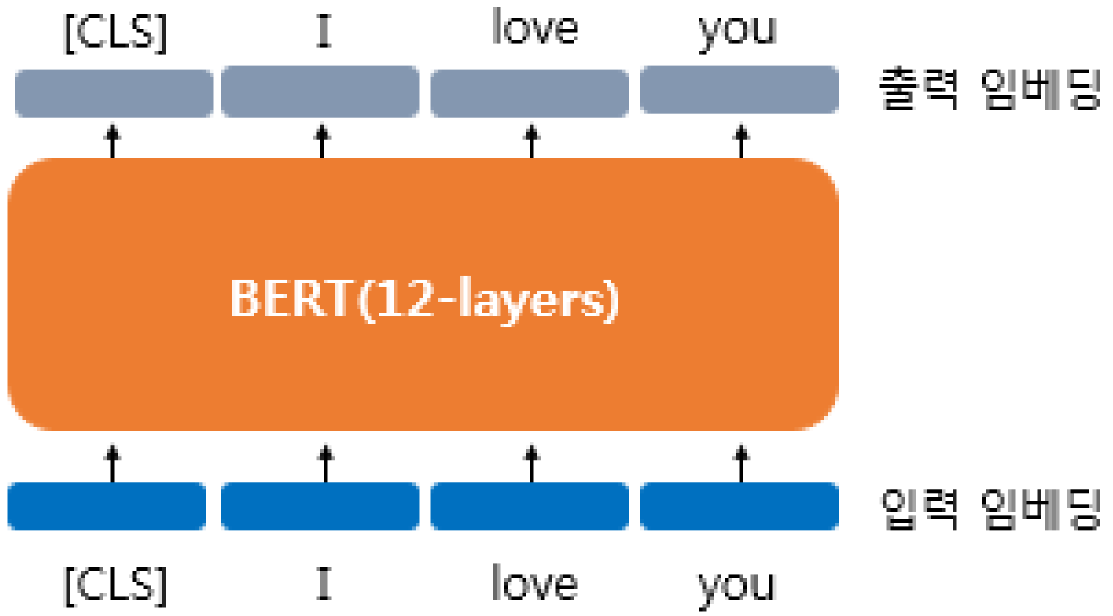
- BERT의 기본 구조는 트랜스포머의 인코더를 쌓아 올린 구조
- Base 버전: 총 12개, Large 버전: 총 24개
- 트랜스포머 인코더 층의 수 = L, d_model의 크기 = D, 셀프 어텐션 헤드의 수 = A

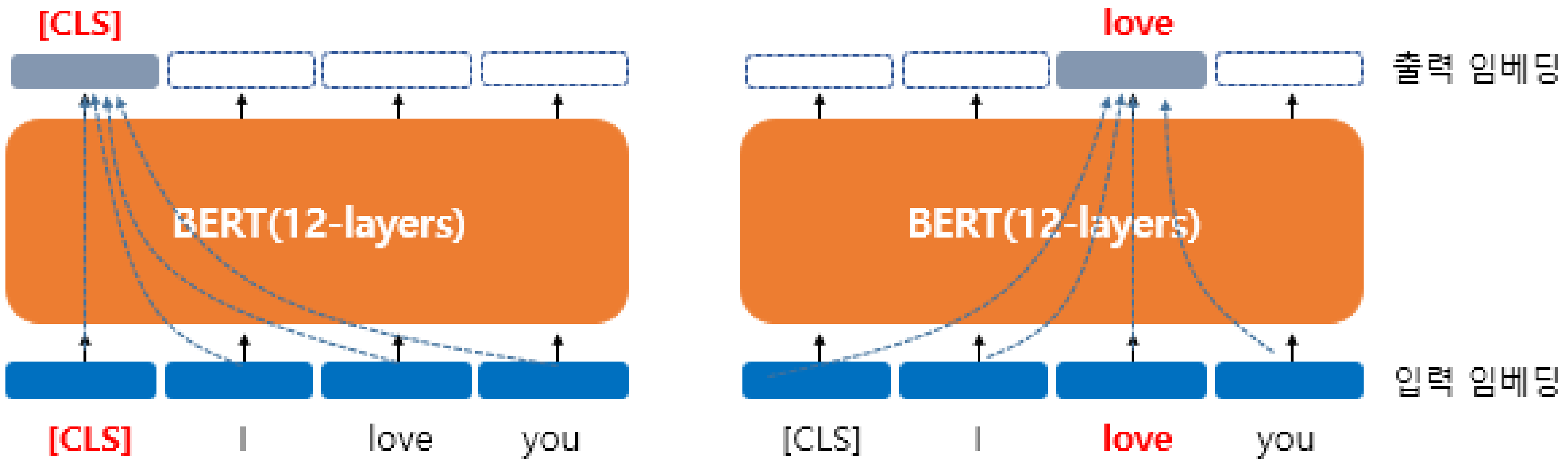
  - BERT-Base : L=12, D=768, A=12 : 110M개의 파라미터
  - BERT-Large : L=24, D=1024, A=16 : 340M개의 파라미터



#03 BERT Architecture

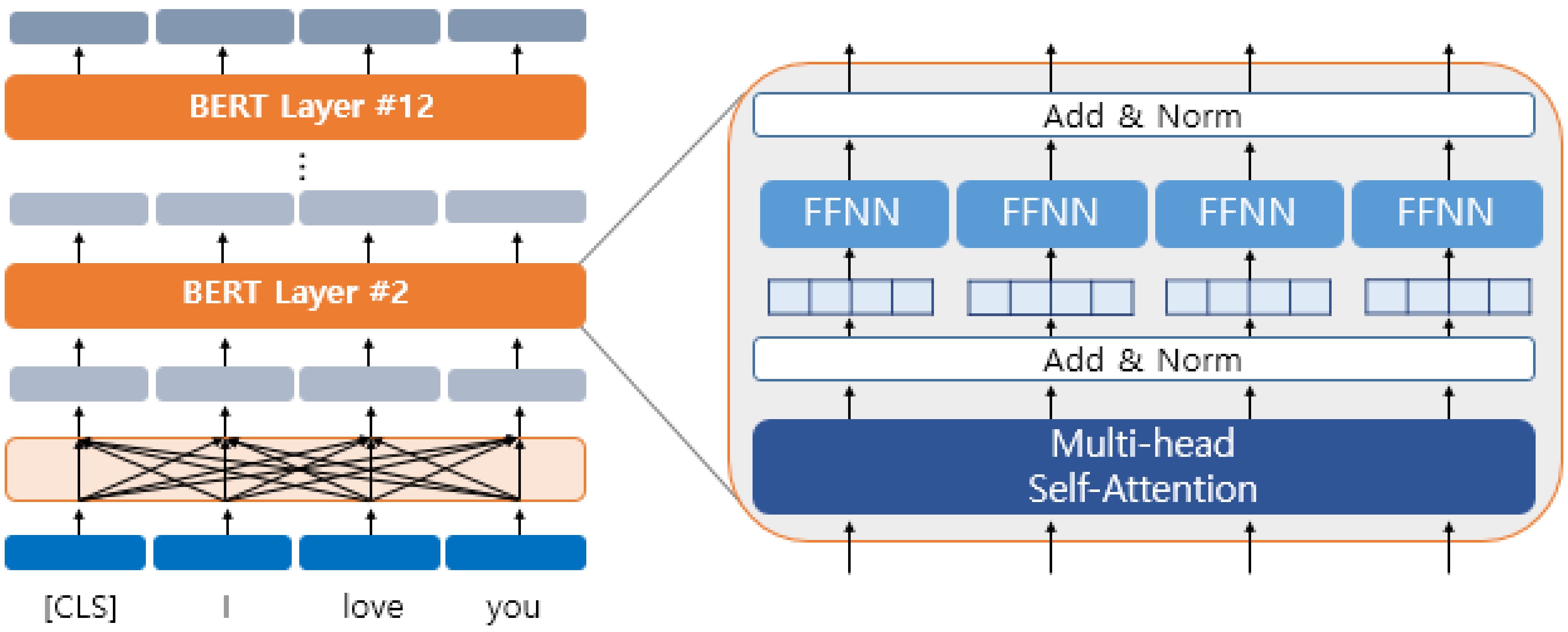
BERT의 문맥을 반영한 임베딩(Contextual Embedding)





#03 BERT Architecture

BERT의 문맥을 반영한 임베딩(Contextual Embedding)

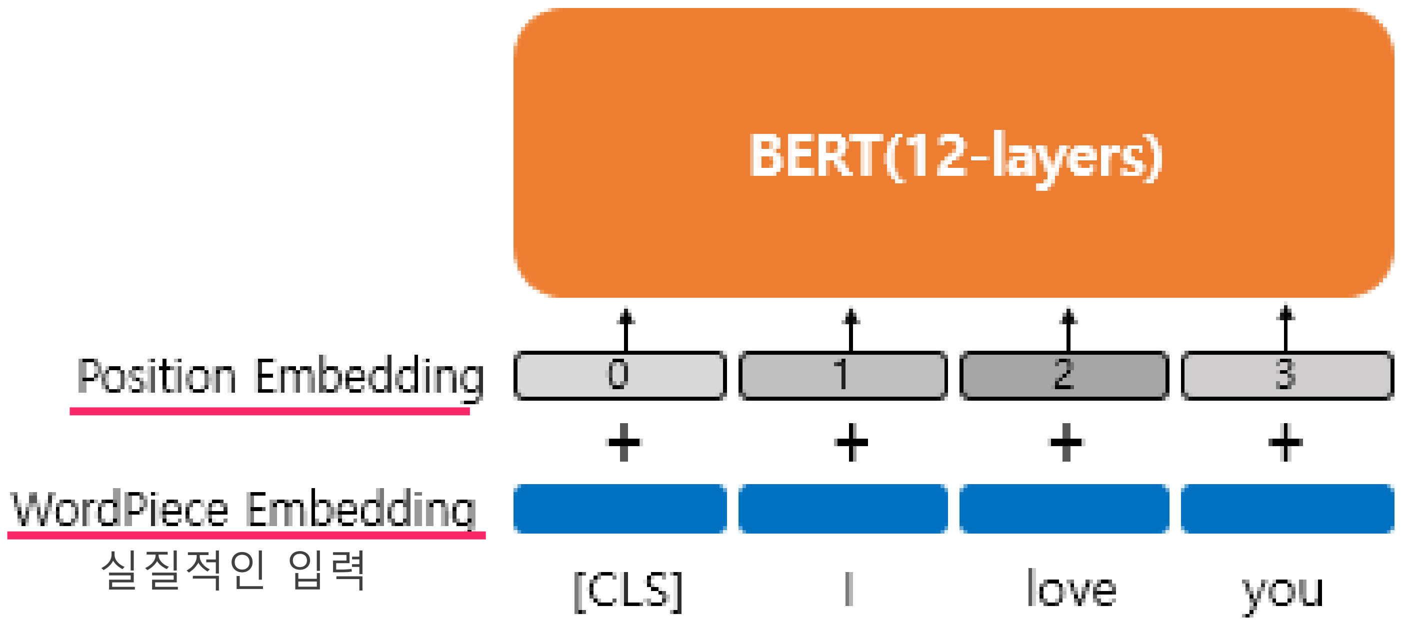


단어보다
더 작은 단위로 쪼개는
Subword Tokenizer
(WordPiece) 사용

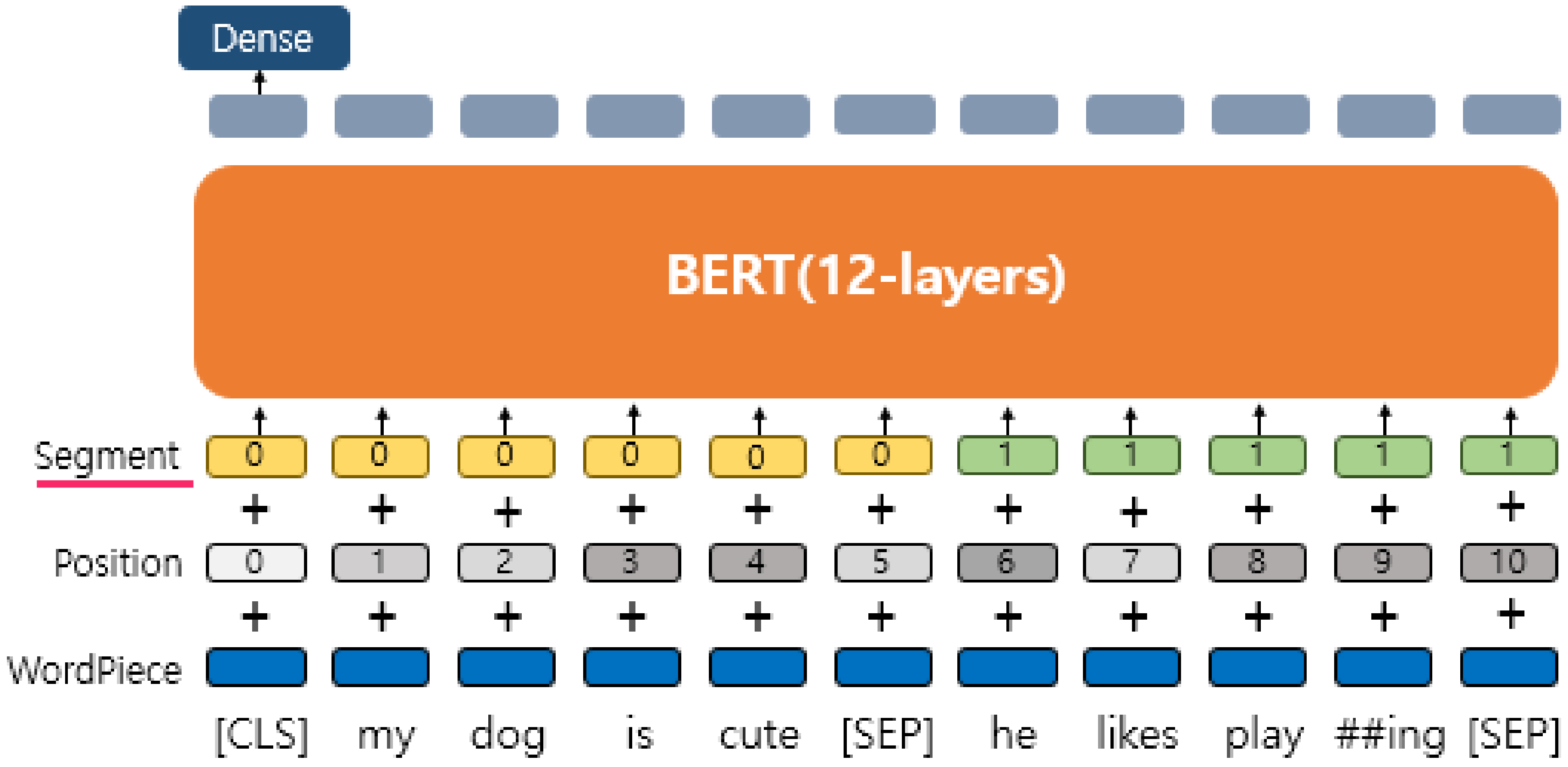
***Subword Tokenizer**
: 기본적으로 자주 등장하는 단어는 그대로 단어 집합에 추가하지만, 자주 등장하지 않는 단어의 경우,
더 작은 단위인 서브워드로 분리되어 서브워드들이 단어 집합에 추가 됨

#03 BERT Architecture

Position Embedding & Segment Embedding



[Position Embedding]



[Segment Embedding]

#03 BERT Architecture

Input/Output Representations

- **Input representation is the sum of**
  - **Token embedding: WordPiece embeddings with a 30,000 token vocabulary**
  - **Segment embedding**
  - **Position embedding**

*sequence: 단일 문장 또는 쌍으로 이루어진 문장

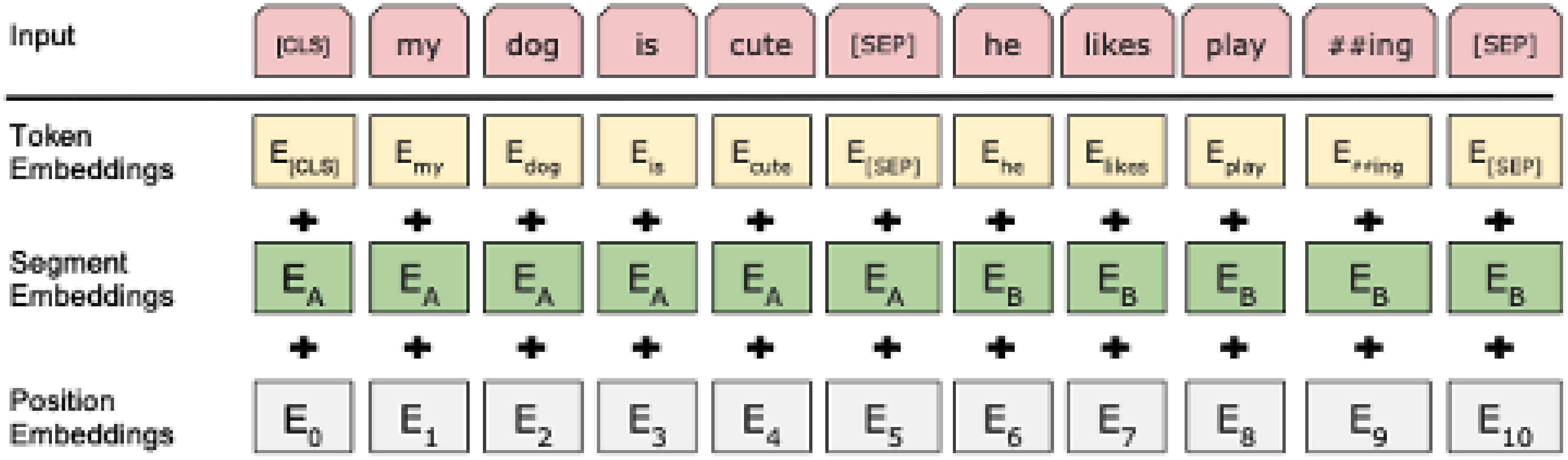
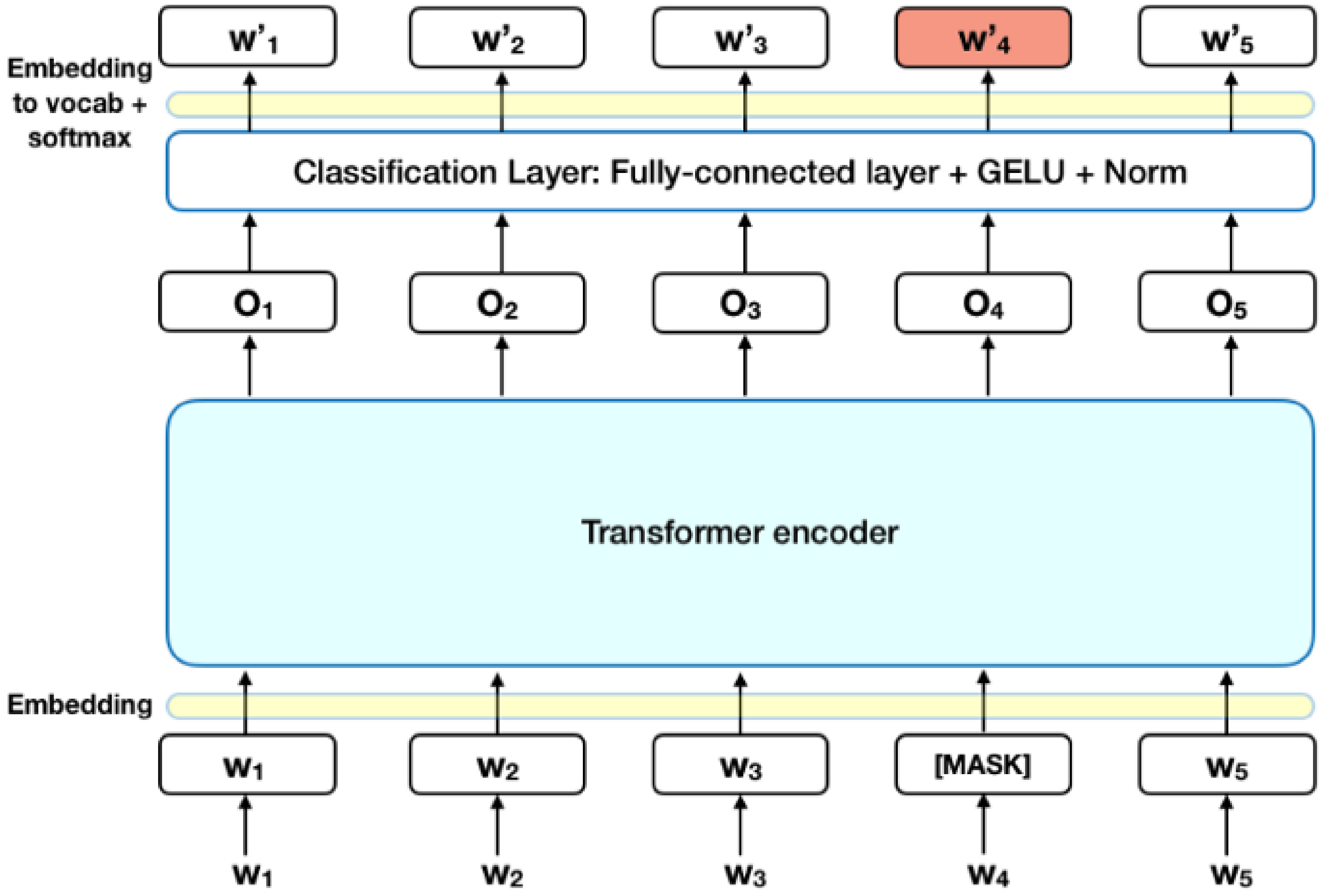


Figure 2: BERT input representation. The input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings.

#03 BERT Architecture

Pre-training

- **Task #1 : Masked LM (MLM)**
  - **15% of each sequence are replaced with a [MASK] token**
  - **Predict the masked words rather than reconstructing the entire input in denoising encoder**



https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
https://www.youtube.com/watch?v=IwtexRHoWG0

#03 BERT Architecture

Pre-training

- **Task #2 : Next Sentence Prediction (NSP)**

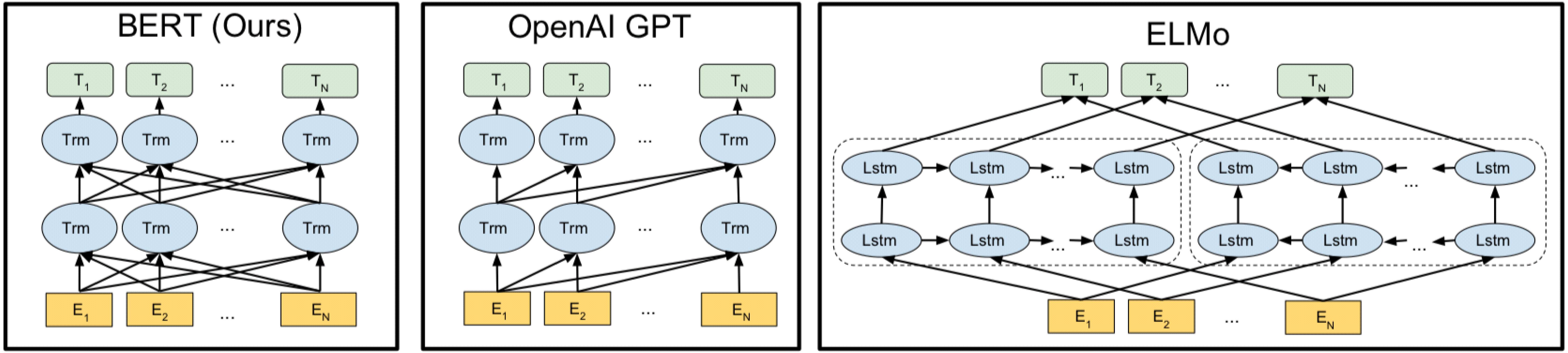
  - Many important downstream tasks such as QA and NLI are based on understanding the relationship between two sentences, which is not directly captured by language modeling
  - A Binarized next sentence prediction task that can be trivially generated from any monolingual corpus is trained
    - 50% of the time B is the actual next sentence that follows A (IsNext)
    - 50% of the time it is a random sentence from the corpus (NotNext)
    - C is used for next sentence prediction
  - Despite its simplicity, pre-training towards this task is very beneficial both QA and NLI

https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270
https://www.youtube.com/watch?v=IwtexRHoWG0

#03 BERT Architecture

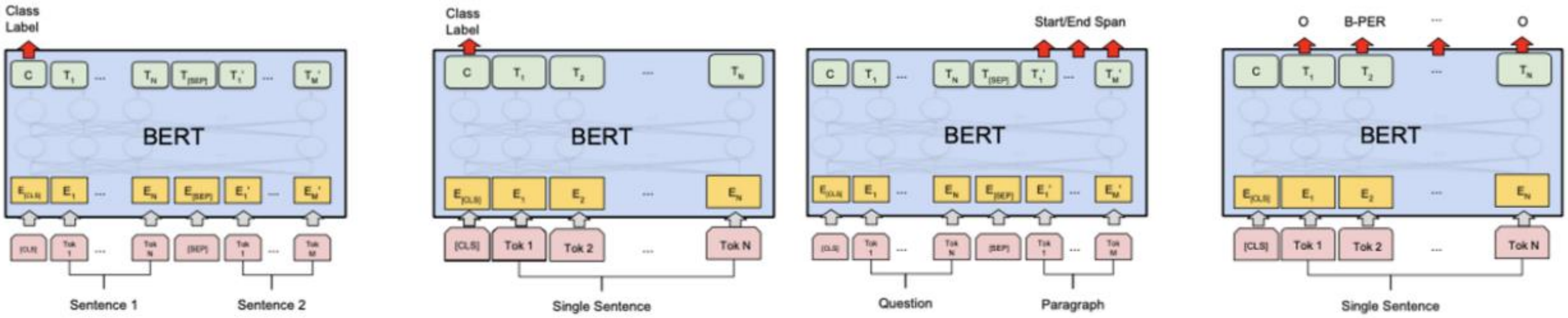
Pre-training

- **Differences in pre-training model architectures**



#03 BERT Architecture

Fine-tuning



(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

Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

#03 BERT Architecture

Fine-tuning

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	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$\text{BERT}_{\text{BASE}}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$\text{BERT}_{\text{LARGE}}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$\text{BERT}_{\text{BASE}}$ (Single)	80.8	88.5	-	-
$\text{BERT}_{\text{LARGE}}$ (Single)	84.1	90.9	-	-
$\text{BERT}_{\text{LARGE}}$ (Ensemble)	85.8	91.8	-	-
$\text{BERT}_{\text{LARGE}}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$\text{BERT}_{\text{LARGE}}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Table 2: SQuAD 1.1 results. The BERT ensemble is 7x systems which use different pre-training checkpoints and fine-tuning seeds.

System	Dev		Test	
	EM	F1	EM	F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	86.3	89.0	86.9	89.5
#1 Single - MIR-MRC (F-Net)	-	-	74.8	78.0
#2 Single - nlnet	-	-	74.2	77.1
Published				
unet (Ensemble)	-	-	71.4	74.9
SLQA+ (Single)	-		71.4	74.4
Ours				
$\text{BERT}_{\text{LARGE}}$ (Single)	78.7	81.9	80.0	83.1

Table 3: SQuAD 2.0 results. We exclude entries that use BERT as one of their components.

#03 RoBERTa

Robustly Optimized BERT Pretraining approach

- BERT가 data, hyperparameter 부분에서 undertrained된 것을 발견
- 기존의 BERT 모델을 유지하면서, 학습 단계의 hyper paramter들을 조정하여 성능을 높이는 방법

- **Dynamic Masking**
  - BERT: pre-training에 사용한 MLM은 무작위로 token에 mask 씌움. 이는 크기가 큰 데이터에 대해 비효율. 즉, 매 학습 단계에서 똑같은 mask를 보게 되는 static masking 방식
  - **RoBERTa**: 매 epoch마다 mask를 새로 씌우는 dynamic masking 사용 <더 나은 성능>
- **Input Format / NSP**
  - BERT: 두 개의 문장을 이어 붙여 input 만듦. 두 문장이 문맥상으로 연결된 문장인지 판단하는 NSP를 pre-training 과정에서 사용
  - **RoBERTa**: NSP 없이 MLM만으로 pre-training. token 수가 512를 넘어가지 않는 선에서 문장을 최대한 이어 붙여서 input을 만듦. <더 나은 성능>
- **Batch Size**
  - 같은 step 수여도 batch size가 클수록 성능이 좋음

Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8
Our reimplementation:			
static	78.3	84.3	92.5
dynamic	78.7	84.0	92.9

Table 1: Comparison between static and dynamic masking for $\text{BERT}_{\text{BASE}}$. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019).

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
Our reimplementation (with NSP loss):				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
Our reimplementation (without NSP loss):				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
$\text{BERT}_{\text{BASE}}$	88.5/76.3	84.3	92.8	64.3
$\text{XLNet}_{\text{BASE}}$ (K = 7)	–/81.3	85.8	92.7	66.1
$\text{XLNet}_{\text{BASE}}$ (K = 6)	–/81.0	85.6	93.4	66.7

Table 2: Development set results for base models pretrained over BOOKCORPUS and WIKIPEDIA. All models are trained for 1M steps with a batch size of 256 sequences. We report F1 for SQuAD and accuracy for MNLI-m, SST-2 and RACE. Reported results are medians over five random initializations (seeds). Results for $\text{BERT}_{\text{BASE}}$ and $\text{XLNet}_{\text{BASE}}$ are from Yang et al. (2019).

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	**3.68**	**85.2**	**92.9**
8K	31K	1e-3	3.77	84.6	92.8

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

#03 RoBERTa

Robustly Optimized BERT Pretraining approach

- **Tokenizer**
  - BERT: 데이터 전처리(Devlin et al., 2019) 이후, 30K 크기의 character-level BPE tokenizer 사용₩
  - **RoBERTa**: 전처리 없이, 50K 크기의 byte-level BPE tokenizer 사용 <비슷한 성능>

- **Data**
  - 데이터 크기가 클수록 성능이 좋아지기 때문에 RoBERTa는 최대한 데이터를 많이 모으는 것에 집중
  - BookCorpus, English Wikipedia, CC-News, OpenWebText, Stories 총 5개의 데이터셋을 합쳐, 총 160GB의 데이터를 완성하였고 BERT-Large보다 <좋은 성능>을 보임

Model	**data**	**bsz**	**steps**	**SQuAD (v1.1/2.0)**	**MNLI-m**	**SST-2**
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	**94.6/89.4**	**90.2**	**96.4**
$\text{BERT}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
$\text{XLNet}_{\text{LARGE}}$						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

Table 4: Development set results for RoBERTa as we pretrain over more data (16GB → 160GB of text) and pretrain for longer (100K → 300K → 500K steps). Each row accumulates improvements from the rows above. RoBERTa matches the architecture and training objective of $\text{BERT}_{\text{LARGE}}$. Results for $\text{BERT}_{\text{LARGE}}$ and $\text{XLNet}_{\text{LARGE}}$ are from Devlin et al. (2019) and Yang et al. (2019), respectively. Complete results on all GLUE tasks can be found in the Appendix.

#03 RoBERTa

Robustly Optimized BERT Pretraining approach

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
Single-task single models on dev										
$BERT_{LARGE}$	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
$XLNet_{LARGE}$	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	**90.2/90.2**	**94.7**	**92.2**	**86.6**	**96.4**	**90.9**	**68.0**	**92.4**	**91.3**	-
Ensembles on test (from leaderboard as of July 25, 2019)										
ALICE	88.2/87.9	95.7	**90.7**	83.5	95.2	92.6	**68.6**	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	**96.8**	**93.0**	67.8	91.6	**90.4**	88.4
RoBERTa	**90.8/90.2**	**98.9**	90.2	**88.2**	96.7	92.3	67.8	**92.2**	89.0	**88.5**

Table 5: Results on GLUE. All results are based on a 24-layer architecture. $BERT_{LARGE}$ and $XLNet_{LARGE}$ results are from Devlin et al. (2019) and Yang et al. (2019), respectively. RoBERTa results on the development set are a median over five runs. RoBERTa results on the test set are ensembles of *single-task* models. For RTE, STS and MRPC we finetune starting from the MNLI model instead of the baseline pretrained model. Averages are obtained from the GLUE leaderboard.

RoBERTa

- 160GB의 데이터
- Dynamic masking
- MLM만으로 pre-train
- Full-sentence 형식의 input
- BERT의 약 32배의 batch size
- byte-level BPE tokenizer

EWHA EURON

#03 KoBERT

Korean BERT (Bidirectional Encoder Representations from Transformers)

- 기존 BERT의 한국어 성능 한계 극복을 위해 개발됨
- 한국어의 불규칙한 언어 변화 특성 반영을 위해 데이터 기반 토큰화 기법을 적용
- 기존 대비 27%의 토큰으로 2.6% 이상의 성능 향상을 보임
- 링 리듀스(ring-reduce)기반 분산 학습 기술 사용
- 파이토치(PyTorch), 텐서플로우(TensorFlow), ONNX, MXNet 등 다양한 딥러닝 API 지원

- 학습셋

데이터	문장	단어
한국어 위키	5M	54M

- 사전(Vocabulary)
  - 크기 : 8,002
  - 한글 위키 기반으로 학습한 토크나이저(SentencePiece)
  - Less number of parameters(92M < 110M)

```
pip install git+https://git@github.com/SKTBrain/KoBERT.git@master

import torch
from kobert import get_pytorch_kobert_model

import mxnet as mx
from kobert import get_mxnet_kobert_model

import onnxruntime
import numpy as np
from kobert import get_onnx_kobert_model
```

공식 깃허브: https://github.com/SKTBrain/KoBERT

#03 Summary

Coronavirus tweets NLP - Text Classification

- ✓ **BERT embedding을 사용한 분석 → 가장 성능이 좋음**

- ✓ **BERT** : 사전 훈련 언어모델
  - ✓ 사전 학습된 대용량의 레이블링 되지 않는(unlabeled) 데이터를 이용하여 언어 모델 (Language Model)을 학습하고 이를 토대로 특정 작업(문서 분류, 질의응답, 번역 등)을 위한 신경망을 추가하는 전이 학습 방법

- ✓ **RoBERTa** : 기존의 BERT 모델을 유지하면서, 학습 단계의 hyper paramter들을 조정하여 성능을 높이는 방법

- ✓ **KoBERT** : 한국어의 불규칙한 언어 변화 특성을 반영한 BERT

THANK YOU