8. Advanced Deep learning model

- BERT, GPT
- Pre-learning , transfer learning

Sogang University Department of Business Administration
Professor Myung Suk Kim

BERT

◆ BERT: Bidirectional Encoder Representations from Transformer (Google , 2018): Encoder only model

- Training the model using the encoder part of the Transformer model (https://wikidocs.net/31379)

[1] Pre-learning with two language tasks :

(1) Masked Language Model :

As a method for learning a language model with bidirectionality, when an input sentence is given, some words are masked to prevent the model from knowing the words, and then predict what the masked words are.-> CBOW extension

Ex) I accessed the _______ account. We play soccer at the bank of ________

(Answer) bank, river

(2) Next Next sentence prediction :

Learning to predict whether two sentences given as input are connected sentences or not (requires prior training with two sentences)

Ex) I acceded the bank account. We play soccer at the bank of the river.

(Answer) not related

[2] Three characteristics

(1) Possible to express word vectors depending on context

Ex) bank has different meanings of ' bank ' and ' riverside ' , and word vector expression suitable for the sentence is possible .

(2) Possible fine tuning in natural language processing tasks (technical characteristics)

(3) Explanation by Attention and Easy to Visualize (Technical Characteristics)

BERT

[3] Usable areas :

- Linguistic acceptability : Distinguishing whether or not an expression makes sense linguistically .
- Natural language reasoning : Distinguishing contradictions in expressions .
- Similarity Prediction : Determines similarity or not .
- Sentiment Analysis : Distinguish between positive and negative .
- Entity Name Recognition : Recognize when an entity is used with a different term (eg English) .
- Machine reading comprehension : Can answer questions about the content when given a sentence .

[4] Utilization of pre-trained BERT model :

- BERT-uncased: all tokens are lowercase
- BERT-cased: No lowercase in tokens . It is used for object name recognition work that needs to preserve case.
- Extract the embedding and use it as a feature extractor
- Text classification, Q&A, etc.

[5] Extraction of embeddings from pre-trained BERT :

- Embedding : A word (last_hidden_states) or sentence (pooler_output) represented as
- Embeddings can be extracted
- The tokenizer used when the model was pretrained (tokenization method) Use
- Requires input pre-processing : [CLS], [SEP], [PAD] tokens to start each sentence , sentence division , sentence length adjustment
- Integer encoding (converting tokens to integers) and attention mask (separating sentences and padding)

GPT

◆ GPT (OpenAI): Decoder only model
- BERT and Similar .
- GPT1, 2: Learning using
- GPT3: sold with exclusive rights to Microsoft (no longer open source)
- ChatGPT (or GPT3.5): no code Open with

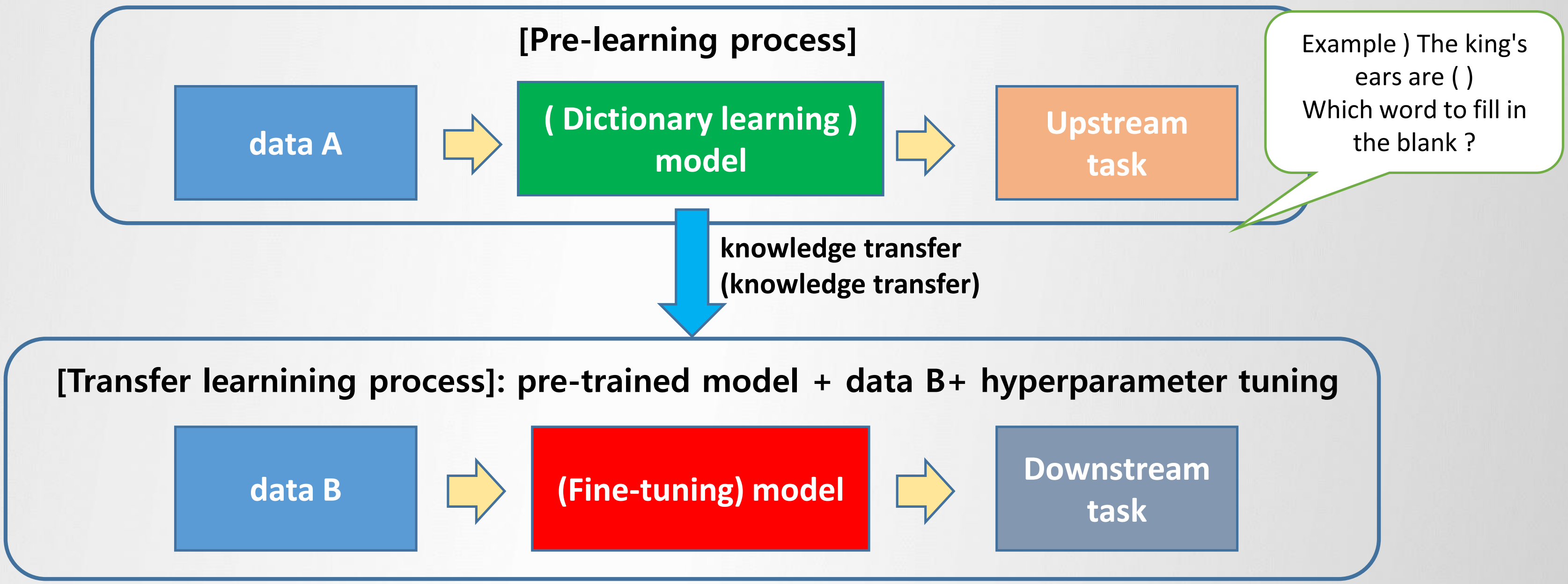
◆ ChatGPT Development history and characteristics
- OpenAI , the developer , released GPT 1 and GPT2 pre-training models
- GPT3 is a size model 100 times larger than GPT2
- The learning cost of GPT3 is about 15 billion won , the pre-training model is made public , and only the API is available
- ChatGPT is called GPT3.5 , and then directly upgraded to GPT4
- ChatGPT has a larger data size than GPT3 and greatly improves performance based on reinforcement learning
- ChatGPT is capable of image input , input text length extension , human -in-the-loop method Wear it to increase sophistication

Transfer learning

◆ Transfer learning

✓ A technique for reusing a model that has learned a specific task to perform other tasks

✓ Model training is faster and tends to perform better than training a new model from scratch



❖ **self-supervised learning** (Self-supervised learning): Among the language models, a model that performs an upstream task such as matching words to fill in the blanks in a sentence is called a mask language model.

It can be performed.Downstream task types

❖ **Downstream task types in natural language tasks**: Document classification (positive-negative), natural language reasoning (true-false), entity name recognition (entity category recognition), question answering, sentence generation

Pre-learning model and transfer learning model

◆ pre-learning model

1. Masked Language Model [fill in the blank]/ Next Next sentence prediction (BERT)

2. Sentence Generation Model (GPT)

◆ transfer learning model

1. Document Classification Model Structure (BERT)

2. Document Pair Classification Model Structure (BERT)

3. Entity Name Recognition Model (BERT)

4. Question Response Model (BERT)

5. Sentence Generation Model (GPT)

BERT Basics

1. Embedding extraction method from pre-trained BERT (using the final encoder layer)

 # Reference : Google BERT standard (Hanbit Media)

```
! pip install transformers

Using Hugging Face 's transformer library and a pre-trained BERT model
from transformers import BertModel , BertTokenizer
import torch

# Entering BertTokenizer.from_pretrained (' model name ') loads the tokenizer that was used when the model was trained
model = BertModel.from_pretrained (' bert -large-uncased')
# BERT- based model with 24 encoders, trained with uncased tokens converted to all lowercase
tokenizer = BertTokenizer.from_pretrained (" bert -large-uncased")
```

BERT Basics

```
#### (1) Input preprocessing ####
sentence = "I love Paris"
tokens = tokenizer. tokenize (sentence)
print(tokens)

tokens = ['[CLS]'] + tokens + ['[SEP]']
print(tokens)

tokens = tokens + ['[PAD]'] + ['[PAD]']
print(tokens)

# If the token is a [PAD] token , set the attention mask to 0 , if it is a normal token , set it to 1
attention_mask = [1 if i !='[PAD]' else 0 for i in tokens]
print( attention_mask )


# convert all tokens to token id
token_ids = tokenizer.convert_tokens_to_ids (tokens)
print( token_ids )

# Convert token_ids and attention_mask to tensors
token_ids = torch.tensor ( token_ids ). unsqueeze (0)
attention_mask = torch.tensor ( attention_mask ). unsqueeze (0)
print( token_ids )
print( attention_mask )
```

BERT Basics

```
#### (2) Embedding extraction ####
# token_ids from the pre-trained BERT model and attention_mask and extract embeddings
model( token_ids , attention_mask = attention_mask )

# Return an output consisting of two values :
# [1] last_hidden_state : representation vector of all tokens from last encoder (24th encoder )
# [2] pooler_output : [CLS] token
last_hidden_state = model(token_ids, attention_mask = attention_mask)[0]
print(last_hidden_state)
pooler_output = model(token_ids, attention_mask = attention_mask)[1]
print(pooler_output)

print(last_hidden_state.shape)  #torch.Size([1, 7, 1024])  # [batch_size, sequence_length, hidden_size]를 의미함
print( pooler_output.shape ) # torch.Size ([1, 1024]) # Means [ batch_size , hidden_size ]

print( last_hidden_state [0][0]) # representation vector of the first token, CLS ( consisting of 1024 elements )
print( last_hidden_state [0][1]) # representation vector of second token I
print( last_hidden_state [0][2]) # expression vector of the third token love
print( last_hidden_state [0][3]) # representation vector of the 4th token Paris
print( len ( last_hidden_state [0][0])) # each representation vector has

print( pooler_output [0]) # You can use
```

BERT Application : Matching Mask Words

1. Google BERT Masked Language Model https://wikidocs.net/153992

```
! pip install transformers

# Masked language model and tokenizer
from transformers import TFBertForMaskedLM
from transformers import AutoTokenizer

# put
Load BERT as a structure for modeling a masked language to match words that say [MASK]
model = TFBertForMaskedLM.from_pretrained (' bert -large-uncased ')
tokenizer = AutoTokenizer.from_pretrained (" bert -large-uncased ")

# Enter BERT : Predict the word corresponding to the position of [MASK]
inputs = tokenizer('Soccer is a really fun [MASK].', return_tensors =' tf ')

# Check the integer encoding result through input_ids in the result converted by tokenizer
print(inputs[' input_ids '])

# Check the segment encoding result that distinguishes sentences through token_type_ids in the tokenizer conversion resu
lt
print(inputs[' token_type_ids '])

# Check the attention mask used to distinguish between real words and padding tokens through attention_mask in the toke
nizer- converted result
print(inputs[' attention_mask '])

# Predict the [MASK] token
from transformers import FillMaskPipeline
pip = FillMaskPipeline (model=model, tokenizer=tokenizer)
```

BERT Application : Matching Mask Words

```
# Print the top 5 candidate words
pip('Soccer is a really fun [MASK].')

pip('The Avengers is a really fun [MASK].')

pip('I went to [MASK] this morning.')
```


[{'score': 0.7621126770973206,
 'token': 4368,
 'token_str': 'sport',
 'sequence': 'soccer is a really fun sport
{'score': 0.20341919362545013,
 'token': 2208,
 'token_str': 'game',
 'sequence': 'soccer is a really fun game
{'score': 0.012208538129925728,
 'token': 2518,
 'token_str': 'thing',
 'sequence': 'soccer is a really fun thing
{'score': 0.0018630228005349636,
 'token': 4023,
 'token_str': 'activity',
 'sequence': 'soccer is a really fun activi
{'score': 0.001335486420430243,
 'token': 2492,
 'token_str': 'field',
 'sequence': 'soccer is a really fun field.

[{'score': 0.2562899589538574,
 'token': 2265,
 'token_str': 'show',
 'sequence': 'the avengers is a really fun sh
{'score': 0.17284104228019714,
 'token': 3185,
 'token_str': 'movie',
 'sequence': 'the avengers is a really fun m
{'score': 0.11107705533504486,
 'token': 2466,
 'token_str': 'story',
 'sequence': 'the avengers is a really fun st
{'score': 0.07248983532190323,
 'token': 2186,
 'token_str': 'series',
 'sequence': 'the avengers is a really fun se
{'score': 0.07046636939048767,
 'token': 2143,
 'token_str': 'film',
 'sequence': 'the avengers is a really fun fil

[{'score': 0.35730698704719543,
 'token': 2147,
 'token_str': 'work',
 'sequence': 'i went to work this morning.'},
{'score': 0.2330448478460312,
 'token': 2793,
 'token_str': 'bed',
 'sequence': 'i went to bed this morning.'},
{'score': 0.1284504383802414,
 'token': 2082,
 'token_str': 'school',
 'sequence': 'i went to school this morning.'},
{'score': 0.062305789440870285,
 'token': 3637,
' token_str ': 'sleep',
'sequence': ' i went to sleep this morning.'},
{'score': 0.04695258289575577,
'token': 2465,
' token_str ': ' class ',
'sequence': ' i went to class this morning.'}]

BERT Application : Matching Mask Words

2. Korean BERT masked language model https://wikidocs.net/152922

```
# Masked language model and tokenizer
from transformers import TFBertForMaskedLM
from transformers import AutoTokenizer
model = TFBertForMaskedLM.from_pretrained ('klue / bert -base ', from_pt =True)
tokenizer = AutoTokenizer.from_pretrained (" klue / bert -base ")

# type BERT
inputs = tokenizer(' Soccer is really fun [ MASK] .', return_tensors =' tf ')

# Check the integer encoding result through input_ids in the result converted by tokenizer
print(inputs[' input_ids '])

# Check the segment encoding result that distinguishes sentences through token_type_ids in the tokenizer conversion resu
lt
print(inputs[' token_type_ids '])

# Check the attention mask used to distinguish between real words and padding tokens through attention_mask in the toke
nizer- converted result
print(inputs[' attention_mask '])

# Predict the [MASK] token
from transformers import FillMaskPipeline
pip = FillMaskPipeline (model=model, tokenizer=tokenizer)
```

BERT Application : Matching Mask Words

```
# Print the top 5 candidate words that can fit in the position of [MASK]
pip(' Soccer is so much fun [ MASK] .')

pip(' Avengers is such a fun [MASK] .')

pip(' I went to work at [MASK] this morning .')
```

```
[{'score': 0.08012557774782181,
'token': 3769,
' token_str ': ' company ',
'sequence': ' I went to work this morning .'},
{'score': 0.06124049797654152,
'token': 1,
' token_str ': '[UNK]',
'sequence': ' I went to work this morning at .'},
{'score': 0.01748666912317276,
'token': 4345,
' token_str ': ' factory ',
'sequence': ' I went to work at the factory this morning .'},
{'score': 0.0161318127065897,
'token': 5841,
' token_str ': ' office ',
'sequence': ' I went to the office this morning .'},
{'score': 0.015360800549387932,
'token': 3671,
' token_str ': ' Seoul ',
'sequence': ' I went to work in Seoul this morning .'}]
```

BERT Application : Determining the adequacy of the following sentence

3. Google BERT 's Predict next sentence **https://wikidocs.net/156767**

!pip install transformers

# (1) Next sentence prediction model and tokenizer
# Since BERT uses a model that has already been trained by someone, the model and tokenizer we use must always have a mapping relationship.
import tensorflow as tf
from transformers import TFBertForNextSentencePrediction
from transformers import AutoTokenizer

# Putting in AutoTokenizer.from_pretrained (' model name ') loads the tokenizer that was used when the model was trained
model = TFBertForNextSentencePrediction.from_pretrained (' bert -base-uncased')
tokenizer = AutoTokenizer.from_pretrained (' bert -base-uncased')

# (2) Input of BERT : Prepare two sentences that actually follow
prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."
next_sentence = "pizza is eaten with the use of a knife and fork. In casual settings, however, it is cut into wedges to be eaten while held in the hand."

# Integer-encode two sentences using the bert -base-uncased tokenizer prepared earlier
encoding = tokenizer(prompt, next_sentence , return_tensors = ' tf ')

Check the integer encoding result through input_ids in the result converted by tokenizer

```
tf.Tensor ( [[ 101 1999 3304 1010 10733 2366 1999 5337 10906 1010 2107 2004 2012 1037 4825 1010
2003 3591 4895 14540 6610 2094 1012 102 10733 2003 8828 2007 1996 2224 1997 1037 5442 1998 9292
1012 1999 10017 10906 1010 2174 1010 2009 2003 3013 2046 17632 2015 2000 2022 8828 2096 2218
1999 1996 2192 1012 102]], shape=(1, 58) , dtype =int32)
```

BERT Application : Determining the adequacy of the following sentence

# Actually output the number of [CLS] token and [SEP] token of the corresponding tokenizer
print(tokenizer.cls_token , ':', tokenizer.cls_token_id)
print(tokenizer.sep_token , ':' , tokenizer.sep_token_id)

# Re-decode the result of the integer encoding above to determine the composition of the current input
print(tokenizer. decode (encoding[' input_ids '][0]))
When two sentences are entered as input in BERT , [CLS] token exists at the beginning ,
When the first sentence ends, a [SEP] token is added , and an additional [SEP] token is added when the second sentence ends.

Check the segment encoding result that distinguishes sentences through token_type_ids in the tokenizer conversion resu

```
tf.Tensor ([[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]], shape=(1, 58) , dtype =int32)
```

# (3) Predict the next sentence : Output the probability value for each label after passing the softmax function
logits = model(encoding[' input_ids '], token_type_ids =encoding[' token_type_ids '])[0]
softmax = tf.keras.layers.Softmax ()

```
tf.Tensor ([[9.9999714e-01 2.8381855e-06]], shape=(1, 2) , dtype =float32)
```

The probability value for index 0 is much greater than the probability value for index 1 . The label predicted by the model means

# Now return the index with the greater probability so that the greater of the two values is the model's predicted value
print(' final prediction label :', tf.math.argmax (probs, axis=-1). numpy ())
# If it is 0 , it means that the two sentences are connected / If it is 1 , it means that the two sentences do not matter

BERT Application : Determining the adequacy of the following sentence

```
# (4) Test two unrelated statements
prompt = "In Italy, pizza served in formal settings, such as at a restaurant, is presented unsliced."
next_sentence = "The sky is blue due to the shorter wavelength of blue light."
encoding = tokenizer(prompt, next_sentence , return_tensors = ' tf ')

# Check the integer encoding result through input_ids in the result converted by tokenizer
print(encoding[' input_ids '])

# Check the integer encoding result through input_ids in the result converted by tokenizer
print(encoding[' input_ids '])

# After passing the softmax function, output
logits = model(encoding[' input_ids '], token_type_ids =encoding[' token_type_ids '])[0]
softmax = tf.keras.layers.Softmax ()
probs = softmax (logits)
print(' final prediction label :', tf.math.argmax (probs, axis=-1). numpy ())
# If it is 0 , it means that the two sentences are connected / If it is 1 , it means that the two sentences do not matter
```

```
Final predicted label : [1]
```

BERT Application : Determining the adequacy of the following sentence

4. Korean BERT 's Next prediction https://wikidocs.net/156774

```
!pip install transformers

# (1) Next sentence prediction model and tokenizer
# If you put TFBertForNextSentencePrediction.from_pretrained ('BERT model name ') then you have two statements
# Load the BERT structure that determines whether the following statements are related
import tensorflow as tf
from transformers import TFBertForNextSentencePrediction
from transformers import AutoTokenizer

# Putting in AutoTokenizer.from_pretrained (' model name ') loads the tokenizer that was used when the model was traine
d
model = TFBertForNextSentencePrediction.from_pretrained (' klue / bert -base', from_pt =True)
tokenizer = AutoTokenizer.from_pretrained (" klue / bert -base")

# (2) Predict the next sentence : the next two sentences
# After passing the softmax function, returns the index with the greater probability of the two values
prompt = "The 2002 World Cup Soccer Tournament was co-hosted with Japan, and is a global event ."
next_sentence = " I went on a trip and Korea's preparations for the 2002 World Cup soccer tournament were perfect ."
encoding = tokenizer(prompt, next_sentence, return_tensors='tf')

logits = model(encoding['input_ids'], token_type_ids=encoding['token_type_ids'])[0]

softmax = tf.keras.layers.Softmax()
probs = softmax(logits)
print('최종 예측 레이블 :', tf.math.argmax(probs, axis=-1).numpy())
# When BERT learns to predict the next sentence, the label of the next two sentences is 0.
```

BERT Application : Determining the adequacy of the following sentence

```
# (3) Test with two disjoint statements
# two unrelated statements
prompt = "The 2002 World Cup Soccer Tournament was co-hosted with Japan, and is a global event ."
next_sentence = " I want to go see a romance movie at the theater "
encoding = tokenizer(prompt, next_sentence, return_tensors='tf')

logits = model(encoding['input_ids'], token_type_ids=encoding['token_type_ids'])[0]

softmax = tf.keras.layers.Softmax()
probs = softmax(logits)
print('최종 예측 레이블 :', tf.math.argmax(probs, axis=-1).numpy())
# When BERT learns to predict the next sentence, the labels of the two practically unrelated sentences are 1
```

BERT Application : Keyword extraction

```
# https://wikidocs.net/159467
1. Keyword extraction using BERT : KeyBERT
! pip install sentence_transformers

# (1) default KeyBERT
import numpy as np
import itertools
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer

doc = """
        Supervised learning is the machine learning task of
        learning a function that maps an input to an output based
        on example input-output pairs.[1] It infers a function
        from labeled training data consisting of a set of
        training examples.[2] In supervised learning, each
        example is a pair consisting of an input object
        (typically a vector) and a desired output value (also
        called the supervisory signal). A supervised learning
        algorithm analyzes the training data and produces an
        inferred function, which can be used for mapping new
        examples. An optimal scenario will allow for the algorithm
        to correctly determine the class labels for unseen
        instances. This requires the learning algorithm to
        generalize from the training data to unseen situations
        in a 'reasonable' way (see inductive bias).
    """
```

BERT Application : Keyword Extraction

```
# Extract words using CountVectorizer
# You can easily extract n-grams by using the n_gram_range argument
# For example , if set to (3, 3), the resulting candidate extracts a trigram that considers three words as a group.
# Extract phrases that are groups of 3 words
n_gram_range = (3, 3)
stop_words = " english "

count = CountVectorizer ( ngram_range = n_gram_range , stop_words = stop_words ).fit([doc])
candidates = count.get_feature_names_out ()

print(' Number of trigrams :', len (candidates))
print(' Print only 5 trigrams :',candidates[:5])
```

```
Output only five trigrams : ['algorithm analyzes training' 'algorithm correctly determine'
'algorithm generalize training' 'allow algorithm correctly' 'analyzes training data']
```

```
# Digitize documents and keywords extracted from documents through SBERT
model = SentenceTransformer (' distilbert -base- nli -mean-tokens')
doc_embedding = model.encode ([doc])
candidate_embeddings = model.encode (candidates)

# Assume that the keywords most similar to the document are good keywords to represent the document
top_n = 5
distances = cosine_similarity ( doc_embedding , candidate_embeddings )
keywords = [candidates[index] for index in distances.argsort ()[0][- top_n :]]
print(keywords)
# 5 keywords output
# Two algorithms to obtain various keywords : Max Sum Similarity, Maximal Marginal Relevance
```

```
['algorithm analyzes training',
'learning algorithm generalize',
'learning machine learning',
'learning algorithm analyzes',
'algorithm generalize training']
```

BERT Application : Keyword Extraction

```
# (2) Max Sum Similarity
# Maximize candidate similarity with document while minimizing similarity between candidates
def max_sum_sim ( doc_embedding , candidate_embeddings , words, top_n , nr_candidates ):
     # Similarity between the document and each keyword
    distances = cosine_similarity ( doc_embedding , candidate_embeddings )

     # Similarity between each keyword
    distances_candidates = cosine_similarity ( candidate_embeddings ,
                                               candidate_embeddings )

     # Pick top_n top_n words among keywords based on cosine similarity .
     words_idx = list( distances.argsort ()[0][- nr_candidates :])
     words_vals = [candidates[index] for index in words_idx ]
     distances_candidates = distances_candidates [ np.ix _( words_idx , words_idx )]

     # Calculate the combination of the least similar keywords among each keyword
    min_sim = np.inf
candidate = None
     for combination in itertools.combinations(range(len(words_idx)), top_n):
          sim = sum([distances_candidates[i][j] for i in combination for j in combination if i != j])
          if sim < min_sim:
               candidate = combination
               min_sim = sim

     return [words_vals[idx] for idx in candidate]
```

BERT Application : Keyword Extraction

```
# Select the top 10 keywords and among these 10 select the 5 least similar to each other
# If you set low nr_candidates, the result is that the 5 keywords printed look very similar to the classic cosine similarity
max_sum_sim ( doc_embedding , candidate_embeddings , candidates, top_n =5, nr_candidates =10)

# relatively high nr_candidates makes 5 more keywords
max_sum_sim ( doc_embedding , candidate_embeddings , candidates, top_n =5, nr_candidates =20)


#(3) Maximal Marginal Relevance
# MMR strives to minimize redundancy and maximize diversity of results in text summarization tasks.
# First select the keywords / key phrases most similar to the document
# Then recursively select new candidates that are similar to the document and not similar to the already selected keyword /
# key phrase
def mmr ( doc_embedding , candidate_embeddings , words, top_n , diversity):

     # A list of similarities between the document and each keyword
    word_doc_similarity = cosine_similarity ( candidate_embeddings , doc_embedding )

     # Similarity between each keyword
    word_similarity = cosine_similarity ( candidate_embeddings )

# Extract the index of the keyword with the highest similarity to the document .
# If document 2 has the highest similarity
    # keywords_idx = [2]
     keywords_idx = [ np.argmax ( word_doc_similarity )]
```

BERT Application : Keyword Extraction

```
# Indexes of documents excluding the index of the keyword with the highest similarity
# If document 2 has the highest similarity
# ==> candidates_idx = [0, 1, 3, 4, 5, 6, 7, 8, 9, 10 ... omitted ...]
candidates_idx = [ i for i in range( len (words)) if i != keywords_idx [0]]

# Repeat below top_n-1 times , since the top keywords have already been extracted .
# ex) If top_n = 5 , the loop below is repeated 4 times .
for _ in range( top_n - 1):
    candidate_similarities = word_doc_similarity [ candidates_idx , :]
    target_similarities = np.max(word_similarity[candidates_idx][:, keywords_idx], axis=1)

    # MMR을 계산
    mmr = (1-diversity) * candidate_similarities - diversity * target_similarities.reshape(-1, 1)
    mmr_idx = candidates_idx[np.argmax(mmr)]

    # keywords & candidates를 업데이트
    keywords_idx.append ( mmr_idx )
    candidates_idx.remove ( mmr_idx )

return [words[ idx ] for idx in keywords_idx ]
```

BERT Application : Keyword Extraction

```
# If you set a relatively low diversity value , the result looks very similar to the conventional cosine similarity only.
mmr ( doc_embedding , candidate_embeddings , candidates, top_n =5, diversity=0.2)
```

```
['algorithm generalize training', 'supervised learning algorithm', 'learning machine learning', 'learning algorithm analyzes',
'learning algorithm generalize']
```

```
# Relatively high diversity value makes 5 diverse keywords
mmr ( doc_embedding , candidate_embeddings , candidates, top_n =5, diversity=0.7)
```

```
['algorithm generalize training', 'labels unseen instances’,
'new examples optimal', 'determine class labels', 'supervised learning algorithm']
```

BERT Application : Keyword Extraction

```
# https://wikidocs.net/159468
2. Keyword extraction using Korean BERT
! pip install sentence_transformers
! pip install konlpy

# (1) 기본 KeyBERT
import numpy as np
import itertools

from konlpy.tag import Okt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sentence_transformers import SentenceTransformer

doc="""
드론 활용 범위도 점차 확대되고 있다. 최근에는 미세먼지 관리에 드론이 활용되고 있다.
서울시는 '미세먼지 계절관리제' 기간인 지난달부터 오는 3월까지 4개월간 드론에 측정장치를 달아 미세먼지 집중 관리를 실시하고 있다.
드론은 산업단지와 사업장 밀집지역을 날아다니며 미세먼지 배출 수치를 점검하고, 현장 모습을 영상으로 담는다.
영상을 통해 미세먼지 방지 시설을 제대로 가동하지 않는 업체와 무허가 시설에 대한 단속이 한층 수월해질 전망이다.
드론 활용에 가장 적극적인 소방청은 광범위하고 복합적인 재난 대응 차원에서 드론과 관련 전문인력 보강을 꾸준히 이어가고 있다.
지난해 말 기준 소방청이 보유한 드론은 총 304대, 드론 조종 자격증을 갖춘 소방대원의 경우 1,860명이다.
이 중 실기평가지도 자격증까지 갖춘 '드론 전문가' 21명도 배치돼 있다.
소방청 관계자는 "소방드론은 재난현장에서 영상정보를 수집, 산악・수난 사고 시 인명수색·구조활동,
유독가스·폭발사고 시 대원안전 확보 등에 활용된다"며
"향후 화재진압, 인명구조 등에도 드론을 활용하기 위해 연구개발(R&D)을 하고 있다"고 말했다.
"""
```

BERT Application : Keyword Extraction

```
doc="""
drones is also gradually expanding . Recently, drones are being used to manage fine dust.
The Seoul Metropolitan Government has been carrying out intensive management of fine dust by attaching measuring
devices to drones for four months from last month to March , the period of the ' fine dust seasonal management
 system'.
The drone flies over industrial complexes and densely populated areas, inspects fine dust emission levels, and
 captures images of the site.
The video is expected to make it easier to crack down on companies that do not properly operate fine dust prevention
facilities and unauthorized facilities.
drones, is steadily reinforcing drones and related experts in the context of wide-ranging and complex disaster response.
As of the end of last year, the Fire Administration had a total of 304 drones and 1,860 firefighters with drone pilot licenses.
Among them, 21' drone experts' who have a practical evaluation guidance certificate are also deployed .
An official from the National Fire Agency said," Firefighting drones collect video information from disaster sites, search
and rescue activities in case of mountain or water accidents,
It is used to secure crew safety in case of toxic gas or explosion accident.”
" We are conducting research and development (R&D) to use drones for firefighting and lifesaving in the future," he said .
"""
```

BERT Application : Keyword Extraction

```
# Create a document with only nouns extracted through the morpheme analyzer
okt = Okt ()

tokenized_doc = okt.pos (doc)
tokenized_nouns = ' '.join([word[0] for word in tokenized_doc if word[1] == 'Noun'])

print(' Part of speech tagging only output 10 :', tokenized_doc [:10])
print(' Extract nouns :', tokenized_nouns )

# Extract words using CountVectorizer of Scikit Learn
# The reason CountVectorizer is used is that n-grams can be easily extracted by using the n_gram_range argument.
# For example , if set to (2, 3), the resulting candidate is a bigram that considers two words as a group and
# Extract a trigram that considers 3 words as a group
n_gram_range = (2, 3)

count = CountVectorizer ( ngram_range = n_gram_range ).fit([ tokenized_nouns ])
candidates = count.get_feature_names_out ()

print(' Number of trigrams :', len (candidates))
print(' Print only 5 trigrams :',candidates[:5])

# Load multilingual SBERT including Korean
model = SentenceTransformer ('sentence-transformers/xlm-r-100langs-bert-base-nli-stsb-mean-tokens')
doc_embedding = model.encode ([doc])
candidate_embeddings = model.encode (candidates)
```

BERT Application : Keyword Extraction

```
# Extract the keywords most similar to the document : Assume that the keywords most similar to the document are
# good keywords to represent the document.
top_n = 5 # print top 5 keywords
distances = cosine_similarity(doc_embedding, candidate_embeddings)
keywords = [candidates[index] for index in distances.argsort()[0][-top_n:]]
print(keywords)

# (2) Max Sum Similarity
def max_sum_sim(doc_embedding, candidate_embeddings, words, top_n, nr_candidates):
   distances = cosine_similarity ( doc_embedding , candidate_embeddings )

# Similarity between each keyword
   distances_candidates = cosine_similarity ( candidate_embeddings , candidate_embeddings )

# Pick top_n top_n words among keywords based on cosine similarity .
    words_idx = list( distances.argsort ()[0][- nr_candidates :])
    words_vals = [candidates[index] for index in words_idx ]
    distances_candidates = distances_candidates [ np.ix _( words_idx , words_idx )]

    # Calculate the combination of the least similar keywords among each keyword
   min_sim = np.inf
candidate = None
    for combination in itertools.combinations(range(len(words_idx)), top_n):
        sim = sum([distances_candidates[i][j] for i in combination for j in combination if i != j])
        if sim < min_sim:
            candidate = combination
            min_sim = sim
    return [words_vals[idx] for idx in candidate]
```

BERT Application : Keyword Extraction

```
# Select the top 10 keywords and among these 10 select the 5 least similar to each other
# With low nr_candidates set, the 5 keywords printed look very similar to the classic cosine similarity only
max_sum_sim ( doc_embedding , candidate_embeddings , candidates, top_n =5, nr_candidates =10)
```

```
['드론 산업 단지', '전망 드론 활용', '드론 산업', '관리 드론 활용', '미세먼지 관리 드론']
```

```
# relatively high nr_candidates makes 5 more keywords
max_sum_sim ( doc_embedding , candidate_embeddings , candidates, top_n =5, nr_candidates =30)
```

```
['소방 드론 재난', '자격증 드론 전문가', '월간 드론 측정', '전망 드론 활용', '미세먼지 관리 드론']
```

(3) Maximal Marginal Relevance

```
def mmr ( doc_embedding , candidate_embeddings , words, top_n , diversity):

    # A list of similarities between the document and each keyword
    word_doc_similarity = cosine_similarity ( candidate_embeddings , doc_embedding )

# Similarity between each keyword
    word_similarity = cosine_similarity ( candidate_embeddings )

# Extract the index of the keyword with the highest similarity to the document .
# If document 2 has the highest similarity
    # keywords_idx = [2]
    keywords_idx = [ np.argmax ( word_doc_similarity )]
```

BERT Application : Keyword Extraction

```
# Indexes of documents excluding the index of the keyword with the highest similarity
# If document 2 has the highest similarity
# ==> candidates_idx = [0, 1, 3, 4, 5, 6, 7, 8, 9, 10 … omitted …]
 candidates_idx = [ i for i in range( len (words)) if i != keywords_idx [0]]

# Repeat below top_n-1 times , since the top keywords have already been extracted .
# ex) If top_n = 5 , the loop below is repeated 4 times .
for _ in range( top_n - 1):
     candidate_similarities = word_doc_similarity [ candidates_idx , :]
     target_similarities = np.max ( word_similarity [ candidates_idx ][:, keywords_idx ], axis=1)

     # Calculate MMR
     mmr = (1-diversity) * candidate_similarities - diversity * target_similarities.reshape(-1, 1)
     mmr_idx = candidates_idx[np.argmax(mmr)]

     # update keywords & candidates
     keywords_idx.append(mmr_idx)
     candidates_idx.remove(mmr_idx)

 return [words[ idx ] for idx in keywords_idx ]
```

BERT Application : Keyword Extraction

If you set a relatively low diversity value , the result looks very similar to the conventional cosine similarity only.
mmr (doc_embedding , candidate_embeddings , candidates, top_n =5, diversity=0.2)

```
['미세먼지 관리 드론',
 '실시 드론 산업',
'관리 드론 활용',
'월간 드론 측정',
'전망 드론 활용']
```

Relatively high diversity value yields
mmr (doc_embedding , candidate_embeddings , candidates, top_n =5, diversity=0.7)

```
['미세먼지 관리 드론',
 '사업 밀집',
'재난 현장 영상',
'산악 수난',
'수치 점검']
```

BERT Application : Document Classification Model

◆ Document classification model structure

- ✓ After tokenizing the input sentence Add special tokens CLS (front) and SEP (back)
- ✓ Input the above contents into the BERT model and extract the pooler output
- ✓ Append additional modules to this vector (probability that the sentence is positive and probability that it is negative)
- ✓ Apply Drop out to Pooler_output
- ✓ Fine tuning : Compare the final output of the model created in this way with the correct answer label, so that the model output matches the correct answer label as closely as possible.

  Update the entire model including task module and BERT layer to be the same

{'sentence': '이 핸폰 밧데리 수명이 정말 길다',
'prediction': '긍정 (positive)',
'positive_data': '긍정 0.5457',
'negative_data': '부정 0.4543',
'positive_width': '54.56999999999999',
'negative_width': '45.43'}

{'sentence': ' this phone The battery life is really long ',
'prediction': ' positive ',
' positive_data ': ' positive 0.5457',
' negative_data ': ' negative 0.4543',
' positive_width ': '54.56999999999999',
' negative_width ': '45.43'}

BERT Application : Document Classification Model

```
### 1. Transfer learning : The goal is to perform a document classification task using a pre-trained language model ###
# Runtime Type : GPU Settings
!pip install ratsnlp
from google.colab import drive
drive.mount ('/content/ gdrvie ')
import os
os.makedirs ("/content/ gdrvie / MyDrive / Colab Notebooks/ Textmining /NLP/ nlpbook /checkpoint- doccls ")

# fine-tuned model (checkpoint) environment settings : kcbert -base model fine-tuned
import torch
from ratsnlp.nlpbook.classification import ClassificationTrainArguments
args = ClassificationTrainArguments(
    pretrained_model_name = "beomi/kcbert-base",
    downstream_corpus_name = "nsmc",  # fine-tuned model 이름
    downstream_model_dir = "/content/gdrvie/MyDrive/Colab Notebooks/Textmining/NLP/nlpbook/checkpoint-doccls",
    batch_size = 32 if torch.cuda.is_available() else 4,
    learning_rate = 5e-5,
    max_seq_length = 128,  # vector length
    epochs = 3,
    tpu_cores = 0 if torch.cuda.is_available () else 8,
seed = 7;
)

# Fixed random seed
from ratsnlp import nlpbook
nlpbook.set_seed ( args )
# Logo settings
nlpbook.set_logger ( args )
```

BERT Application : Document Classification Model

```python
# Download corpus
from Korpora import Korpora
Korpora.fetch(
    corpus_name = args.downstream_corpus_name,
    root_dir = args.downstream_corpus_root_dir,
    force_download = True,
)

# Tokenizer
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained(
    args.pretrained_model_name,
    do_lower_case = False,
)
```

```
Korpora
  nsmc
    cached_test_BertT
    cached_test_BertT
    cached_train_BertT
    cached_train_BertT
    ratings_test.txt
    ratings_train.txt
```

ratings_train.txt ×

```
id	document	label
9976970 아 더빙.. 진짜 짜증나네요 목소리	0
3819312 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구
10265843	너무재밓었다그래서보는것을추천한다	0
9045019 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정 0
6483659 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨어
5403919 막 걸음마 뗀 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋ
7797314 원작의 긴장감을 제대로 살려내지못했다.	0
9443947 별 반개도 아깝다 욕나온다 이응경 길용우 연기생활이몇년
7156791 액션이 없는데도 재미 있는 몇안되는 영화 1
```

BERT Application : Document Classification Model

```
# data pre-processing
# Building the training dataset
from ratsnlp.nlpbook.classification import NsmcCorpus , ClassificationDataset
corpus = NsmcCorpus ()
train_dataset = ClassificationDataset (
    args = args ,
corpus = corpus,
tokenizer = tokenizer ,
mode = "train";
)
```

```
ratsnlp:*** Example ***
ratsnlp:sentence: 아 더빙.. 진짜 짜증나네요 목소리
ratsnlp:tokens: [CLS] 아 더 ##빙 . . 진짜 짜증나네 ##요 목소리 [SEP] [PA
ratsnlp:label: 0
ratsnlp:features: ClassificationFeatures(input_ids=[2, 2170, 832, 5045,
ratsnlp:*** Example ***
ratsnlp:sentence: 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구
ratsnlp:tokens: [CLS] 흠 . . . 포 ##스터 ##보고 초딩 ##영화 ##줄 . . . .
ratsnlp:label: 1
ratsnlp:features: ClassificationFeatures(input_ids=[2, 3521, 17, 17, 17,
ratsnlp:*** Example ***
ratsnlp:sentence: 너무재밓었다그래서보는것을추천한다
ratsnlp:tokens: [CLS] 너무 ##재 ##밓 ##었다 ##그래 ##서 ##보는 ##것을 ##
ratsnlp:label: 0
ratsnlp:features: ClassificationFeatures(input_ids=[2, 8069, 4089, 7847,
```

BERT Application : Document Classification Model

- ClassificationFeatures

  Contains the following 4 elements

(1) input_ids (by index Converted Token Sequence)

(2) attention_mask (distinguishes whether that token is padding (0) or not (1))

(3) token_type_ids (segment information)

(4) contains 4 elements

- token_type_ids

the BERT model, segment information is such that the token sequence of the first document is 0 and the token sequence of the second document is binary classification is form . This document is a task to input one movie review document and classify the polarity. Segment information is all 0 .

```
train_dataset [1]
len ( train_dataset [1] .input_ids  #128
```

```
ClassificationFeatures(input_ids=[2, 3521, 17, 17, 17, 3294, 13069, 8190, 10635, 13796, 4006, 17, 17, 17, 17, 17613
2175, 8030, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], attention_mask=[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
token_type_ids=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], label=1)
```

BERT Application : Document Classification Model

```
# Build a training data loader : instance from train_dataset batch_size Creating a batch after non-repair random extraction
# as many as the number ( collate_fn )
from torch.utils.data import DataLoader , RandomSampler
train_dataloader = DataLoader (
    train_dataset ,
    batch_size = args. batch_size ,
    sampler = RandomSampler ( train_dataset , replacement = False),
    collate_fn = nlpbook. data_collator ,
    drop_last = False,
    num_workers = args. cpu_workers ,
)

# Build the data loader for evaluation : set the instance to batch_size Extracted in order by number
from torch.utils.data import SequentialSampler
val_dataset = ClassificationDataset(
    args = args,
    corpus = corpus,
    tokenizer = tokenizer,
    mode = "test",
)
val_dataloader = DataLoader(
    val_dataset,
    batch_size = args.batch_size,
sampler = SequentialSampler ( val_dataset ),
    collate_fn = nlpbook.data_collator ,
    drop_last = False,
    num_workers = args. cpu_workers ,
)
```

BERT Application : Document Classification Model

```
# load the model
# Initialize the model
from transformers import BertConfig, BertForSequenceClassification
pretrained_model_config = BertConfig.from_pretrained(
    args.pretrained_model_name,
    num_labels = corpus.num_labels,
)
    model = BertForSequenceClassification.from_pretrained(
    args.pretrained_model_name,
    config = pretrained_model_config,
)

# 모델 학습:
# TASK definition : ClassificationTask uses Adam as optimizer and ExponentialLR as running rate scheduler . use
from ratsnlp.nlpbook.classification import ClassificationTask
task = ClassificationTask ( model , args )

# trainer definition
trainer = nlpbook.get_trainer ( args )

# Start learning : Use the fine-tuned model (checkpoint) created in the downstream_model_dir ( takes more than 2 hour
s )
trainer. fit (
task,
    train_dataloaders = train_dataloader ,
    val_dataloaders = val_dataloader ,
) #ckpt _ Can be used if at least one extension file is created
```

BERT Application : Document Classification Model

```
####### 2. Inference: Movie review sentiment analysis web service ##########
# Inference settings
from ratsnlp.nlpbook.classification import ClassificationDeployArguments
args = ClassificationDeployArguments (
    pretrained_model_name = " beomi / kcbert -base",
    downstream_model_dir = "/content/ gdrvie / MyDrive / Colab Notebooks/ Textmining /NLP/ nlpbook /checkpoint-
doccls ",
    max_seq_length = 128;
)

# Load tokenizer and model
# load the tokenizer
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained (
    args.pretrained_model_name ,
    do_lower_case = False,
)

# load checkpoints (fine-tuned model load)
import torch
fine_tuned_model_ckpt = torch.load (
    args.downstream_model_checkpoint_fpath ,
    map_location = torch.device (" cpu "),
)
```

BERT Application : Document Classification Model

```
# BERT setting load
from transformers import BertConfig
pretrained_model_config = BertConfig.from_pretrained(
    args.pretrained_model_name,
    num_labels = fine_tuned_model_ckpt["state_dict"]["model.classifier.bias"].shape.numel(),
)

# BERT model initialize
from transformers import BertForSequenceClassification
model = BertForSequenceClassification ( pretrained_model_config )

# Inject checkpoint into initialized BERT model
model.load_state_dict ({ k.replace ("model.",""): v for k, v in fine_tuned_model_ckpt [' state_dict '].items()})

# switch to evaluation mode
model.eval ()
```

BERT Application : Document Classification Model

```
from pytorch_lightning.callbacks import prediction_writer
# Infernce function: After tokenizing sentences, make input_ids, attention_masks, token_type_ids
def inference_fn(sentence):
  inputs = tokenizer(
      [sentence],
      max_length = args.max_seq_length,
      padding = "max_length",
      truncation = True,
  )
  with torch.no_grad():
outputs = model(**{k: torch.tensor (v) for k, v in inputs.items ()}) # inputs to pytorch convert
prob = outputs.logits.softmax (dim=1)                # to logits Apply
    positive_prob = round(prob[0][1].item(), 4)      # round to 4 decimal places
    negative_prob = round(prob[0][0].item(), 4)
    pred = " positive " if torch.argmax (prob) == 1 else " negative "
    # Pred according to the location of the maximum value of the predicted probability to make
return {
'sentence': sentence,
'prediction': pred ,
' positive_data ': f" positive { positive_prob }",
' negative_data ': f" negative { negative_prob }",
' positive_width ': f"{ positive_prob * 100}",
' negative_width ': f"{ negative_prob * 100}",
}

sentence1 = ' This phone The battery life is really long
inference_fn (sentence1)
```

```
{'sentence': '이 핸폰 밧데리 수명이 정말 길다',
 'prediction': '긍정 (positive)',
 'positive_data': '긍정 0.5457',
 'negative_data': '부정 0.4543',
 'positive_width': '54.56999999999999',
 'negative_width': '45.43'}
```

```
{'sentence': ' this phone The battery life is really long '
```

BERT application : question-answering model

◆ Q&A model

✓ Finding answers to questions in text

yes)

지문	숭례문은 조선의 수도였던 서울의 4대문 중의 하나로 남쪽의 대문이다. 흔히 남대문이라고도 부른다. 서울 4대문 및 보신각의 이름은 오행사상을 따라 지어졌는데, 이런 명칭은 인, 의, 례, 지, 신의 5덕을 표현한 것이었으며, 숭례문의 '례'는 여기서 유래한 것이다. 숭례문의 편액은 지봉유설에 따르면 양녕대군이 썼다고 알려져 있으나 이설이 많다. 1396년(태조 5년)에 최유경의 지휘로 축성하였다. 1447년(세종 29년)과 1479년(성종 10년) 고쳐 지었다.
질문	숭례문이 축성된 연도는?
답변	1396년

✓ The model's inputs are questions and fingerprints ; The output is [(probability of being the start of the correct answer), (probability of being the end of the correct answer)] for each token.

입력	숭례문이 축성된 연도는? 숭례문은 조선의 수도였던 서울의 4대문 중의 하나로 남쪽의 대문이다. 흔히 남대문이라고도 부른다. 서울 4대문 및 보신각의 이름은 오행사상을 따라 지어졌는데, 이런 명칭은 인, 의, 례, 지, 신의 5덕을 표현한 것이었으며, 숭례문의 '례'는 여기서 유래한 것이다. 숭례문의 편액은 지봉유설에 따르면 양녕대군이 썼다고 알려져 있으나 이설이 많다. 1396년(태조 5년)에 최유경의 지휘로 축성하였다. 1447년(세종 29년)과 1479년(성종 10년) 고쳐 지었다.	출력	... 1396 ->[0.93 , 0.01] years ->[0.01, 0.90] ...

◆ Q&A model structure

✓ After tokenizing the question and fingerprint sentences, connect them in the form of

BERT application : question-answering model

Q&A model

Fingerprint	**Sungnyemun is one of the four gates of Seoul, the capital of Joseon. One of them is the southern gate . It is also often called Namdaemun . The names of Seoul's four main gates and Bosingak Pavilion were named after the five elements , and these na mes expressed the five virtues of benevolence , courtesy , courtesy , knowledge , and godliness , and the ' Rye ' of Sungnyem un was derived from them . According to the Jibong Yuseol, the tablet of Sungnyemun was built by Yangnyeongdaegun. It is s aid to have been written , but there are many heresies . It was built in 1396 (the 5th year of King Taejo) under the direction of Yougyeong Choi . It was rebuilt in 1447 (29th year of King Sejong) and 1479 (10th year of King Seongjong) .**
question	In what year was Sungnyemun Gate built ?
answer	1396 _

input	In what year was Sungnyemun Gate built ? Sungnyemun is one of the four gates of Seoul, the capital of Joseon. One of them is the southern gat e . It is also often called Namdaemun . The names of Seoul's four main gates and Bosingak Pavilion w ere named after the five elements , and these names expressed the five virtues of benevolence , cou rtesy , courtesy , knowledge , and godliness , and the ' Rye ' of Sungnyemun was derived from them . According to the Jibong Yuseol, the tablet of Sungnyemun was built by Yangnyeongdaegun. It is said t o have been written , but there are many heresies . It was built in 1396 (the 5th year of King Taejo) under the direction of Yougyeong Choi . It was rebuilt in 1447 (29th year of King Sejong) and 1479 (10th year of King Seongjong) .	Print	… 1396 ->[0.93 , 0.01] years ->[0.01, 0.90] …

BERT application : question-answering model

```
########### 1. Transfer learning : training a question-answer model
!pip install ratsnlp

from google.colab import drive
drive.mount ('/content/ gdrvie ')

import os
os.makedirs("/content/gdrvie/MyDrive/Colab Notebooks/Textmining/NLP/nlpbook/checkpoint-qa")

# model environment setting
import torch
from ratsnlp.nlpbook.qa import QATrainArguments
args = QATrainArguments(
    pretrained_model_name = "beomi/kcbert-base",
    downstream_corpus_name = "korquad-v1",
    downstream_model_dir = "/content/gdrvie/MyDrive/Colab Notebooks/Textmining/NLP/nlpbook/checkpoint-qa",
    batch_size = 32 if torch.cuda.is_available() else 4,
    learning_rate = 5e-5,
    max_seq_length = 128,
    max_query_length = 32,
    doc_stride = 64,
    epochs = 3,
    tpu_cores = 0 if torch.cuda.is_available () else 8,
    seed = 7;
)
```

BERT application : question-answering model

```
# Fixed random seed
from ratsnlp import nlpbook
nlpbook.set_seed ( args )

# Logo settings
nlpbook.set_logger ( args )

# Download corpus
nlpbook.download_downstream_dataset ( args )

# prepare the tokenizer
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained (
    args.pretrained_model_name ,
    do_lower_case = False,
)

# data preprocessing
# Building the training dataset
from ratsnlp.nlpbook.qa import KorQuADV1Corpus, QADataset
corpus = KorQuADV1Corpus()
train_dataset = QADataset (
    args = args ,
corpus = corpus,
tokenizer = tokenizer,
mode = "train";
)
```

BERT application : question-answering model

```
# Build a training data loader
from torch.utils.data import DataLoader , RandomSampler
train_dataloader = DataLoader(
    train_dataset,
    batch_size = args.batch_size,
    sampler = RandomSampler(train_dataset, replacement = False),
    collate_fn = nlpbook.data_collator,
    drop_last = False,
    num_workers = args.cpu_workers,
)

# Building a data loader for evaluation
from torch.utils.data import SequentialSampler
val_dataset = QADataset (
    args = args ,
corpus = corpus,
tokenizer = tokenizer,
mode = " val ",
)

val_dataloader = DataLoader (
    val_dataset ,
    batch_size = args. batch_size ,
sampler = SequentialSampler ( val_dataset ),
    collate_fn = nlpbook.data_collator ,
    drop_last = False,
    num_workers = args. cpu_workers ,
)
```

BERT application : question-answering model

```
# load the model
# model initialization
from transformers import BertConfig, BertForQuestionAnswering
pretrained_model_config = BertConfig.from_pretrained(
    args.pretrained_model_name,
)
model = BertForQuestionAnswering.from_pretrained(
    args.pretrained_model_name,
    config = pretrained_model_config,
)

# 모델 학습
# TASK 정의
from ratsnlp.nlpbook.qa import QATask
task = QATask (model, args )

# trainer definition
trainer = nlpbook.get_trainer ( args )

# start learning
trainer. fit (
task,
    train_dataloaders = train_dataloader ,
    val_dataloaders = val_dataloader ,
)
```

BERT application : question-answering model

```
###### 2. Inference: Make Q&A model web service
# Inference settings
from ratsnlp.nlpbook.qa import QADeployArguments
args = QADeployArguments (
    pretrained_model_name = " beomi / kcbert -base",
    downstream_model_dir = "/content/ gdrvie / MyDrive / Colab Notebooks/ Textmining /NLP/ nlpbook /checkpoint-
qa ",
    max_seq_length = 128;
    max_query_length = 32;
)

# Load tokenizer and model
# load the tokenizer
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained (
    args.pretrained_model_name ,
    do_lower_case = False,
)

# load checkpoint
import torch
fine_tuned_model_ckpt = torch.load (
    args.downstream_model_checkpoint_fpath ,
    map_location = torch.device (" cpu "),
)
```

BERT application : question-answering model

```
load BERT settings
from transformers import BertConfig
pretrained_model_config = BertConfig.from_pretrained (
    args.pretrained_model_name ,
)

# Initialize the BERT model
from transformers import BertForQuestionAnswering
model = BertForQuestionAnswering ( pretrained_model_config )

# Inject checkpoint
model.load_state_dict({k.replace("model.",""): v for k, v in fine_tuned_model_ckpt['state_dict'].items()})

# Convert to evaluation mode
model.eval()

from pytorch_lightning.callbacks import prediction_writer

# Inference function
def inference_fn(question, context):

  if question and context:
    truncated_query = tokenizer.encode(
        question,
        add_special_tokens = False,
        truncation = True,
        max_length = args.max_query_length
    )
```

BERT application : question-answering model

```
  inputs = tokenizer.encode_plus(
      text = truncated_query,
      text_pair = context,
      truncation = "only_second",
      padding = "max_length",
      max_length = args.max_seq_length,
      return_token_type_ids = True,
  )

  with torch.no_grad():
    outputs = model(**{k: torch.tensor([v]) for k, v in inputs.items()})
    start_pred = outputs.start_logits.argmax(dim=-1).item()
    end_pred = outputs.end_logits.argmax(dim=-1).item()
    pred_text = tokenizer.decode(inputs['input_ids'][start_pred:end_pred+1])

else:
  pred_text = ""
return {
    'question': question,
    'context': context,
    'answer': pred_text,
}
```

BERT application : question-answering model

```
question = '서강대 AI MBA는 언제 개설되었나요?'
context = '서강대 AI MBA는 2020년에 개설되었으며, 인공지능 빅데이터 기술을 겸비한 비즈니스 역량을 갖춘 인재 양성에 주력하고 있다. 2023년 현재 4기 신입생을 선발하였고, 졸업생은 2024년 기준으로 120여명을 넘을 예정이다.'
inference_fn(question, context)
```

```
{'question': '서강대 AI MBA는 언제 개설되었나요?',

'context': '서강대 AI MBA는 2020년에 개설되었으며, 인공지능 빅데이터 기술을 겸비한 비즈니스 역량을 갖춘 인재 양성에 주력하고 있다. 2023년 현재 4기 신입생을 선발하였고, 졸업생은 2024년 기준으로 120여명을 넘을 예정이다.',

'answer': '2020년'}
```

```
question = ' When was the Sogang University AI MBA opened ?'
context = ' Sogang University's AI MBA was established in 2020 and focuses on cultivating talents with business capabilities equipped with artificial intelligence and big data technology . As of 2023 , the 4th class of freshmen has been selected , and the number of graduates is expected to exceed 120 as of 2024. '
inference_fn (question, context)
```

```
{'question': ' When was the Sogang University AI MBA opened ?',

'context': ' Sogang University's AI MBA was established in 2020 and focuses on cultivating talents with business capabilities equipped with artificial intelligence and big data technology . As of 2023 , the 4th class of freshmen has been selected , and the number of graduates is expected to exceed 120 as of 2024. ' ,

'answer': ' year 2020 '}
```