10장 | 자연어 처리를 위한 임베딩

WEEK 16주차

10.3 한국어 임베딩

코드 10-53 문장의 토크나이징

```
text = "나는 파이토치를 이용한 딥러닝을 학습중이다."
marked_text = "[CLS] " + text + " [SEP]" # 문장의 시작은 [CLS], 문장의 끝은 [SEP]로 형태를 맞춥니다
tokenized_text = tokenizer.tokenize(marked_text) # 사전 훈련된 버트 토크나이저를 이용해서 문장을 단어로 쪼갭니다
print(tokenized_text)
```

"나는 파이토치를 이용한 딥러닝을 학습중이다."라는 문장을 쪼갠(토크나이징한) 결과 : **['[CLS]', '나는', '파', '##이', '##토', '##치를', '이', '##용한', '딥', '##러', '##닝', '##을', '학', '##습', '##중', '##이다', '.', '[SEP]']**

쪼개진 단어들이 정확하지 않음 (과하다 싶을 정도로 잘게 쪼개져 있음)
→ 버트 토크나이저가
단어의 가장 작은 조각을 기준으로 쪼개도록 설계되었기 때문

코드 10-54 모델을 훈련시킬 텍스트 정의

```
text = "과수원에 사과가 많았다." \
       "친구가 나에게 사과했다."\
       "백설공주는 독이 든 사과를 먹었다."

marked_text = "[CLS] " + text + " [SEP]"
tokenized_text = tokenizer.tokenize(marked_text) # 문장을 토큰으로 분리
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text) # 토큰 문자열에 인덱스를 매핑

for tup in zip(tokenized_text, indexed_tokens): # 단어와 인덱스를 출력
    print('{:<12} {:>6,}'.format(tup[0], tup[1]))
```

[CLS] 101
과 8,898
##수 15,891
##원에 108,280
사 9,405
##과 11,882
##가 11,287
많 9,249
##았다 27,303
. 119
친 9,781
##구 17,196
##가 11,287
나 8,982
##에게 26,212
사 9,405
##과 11,882
##했다 12,490

버트는 문장을 구별하기 위해 1과 0을 사용

→ 문장이 바뀔 때마다 0에서 1로, 1에서 0으로 바뀜

⇒ [0, 0, 1, 1, 1, 0, 0, 0] 이라는 결과
= 세 개의 문장으로 구성되었음을 의미

. 119
백 9,331
##설 31,928
##공 28,000
##주는 100,633
독 9,088
##이 10,739
든 9,115
사 9,405
##과 11,882
##를 11,513
먹 9,266
##었다 17,706
. 119
[SEP] 102

코드 10-57 모델 생성

```
model = BertModel.from_pretrained('bert-base-multilingual-cased',
                                  output_hidden_states = True,)

model.eval()
```

from_pretrained를 사용하여 인터넷에서 사전 훈련된 모델 내려받음

'bert-base-multilingual-cased' 모델은 12개의 계층(임베딩 계층을 포함한다면 13개의 계층)으로 구성된 심층 신경망

ⓐ `'bert-base-multilingual-cased'` : 영어 외에 다국어에 대한 임베딩 처리를 할 때 사용하는 모델

ⓑ `output_hidden_states` : 버트 모델에서 은닉 상태의 값을 가져오기 위해 사용

코드 10-58 모델의 은닉 상태 정보 확인

```
print ("계층 수:", len(hidden_states), "  (initial embeddings + 12 BERT layers)")
layer_i = 0

print ("배치 수:", len(hidden_states[layer_i]))
batch_i = 0

print ("토큰 수:", len(hidden_states[layer_i][batch_i]))
token_i = 0

print ("은닉층 유닛 수:", len(hidden_states[layer_i][batch_i][token_i]))
```

계층 수: 13 (initial embeddings + 12 BERT layers)
배치 수: 1
토큰 수: 33
은닉층 유닛 수: 768

코드 10-63 텐서 차원 변경

```
token_embeddings = token_embeddings.permute(1,0,2)
token_embeddings.size()
```

`permute` 는 `transpose` 와 유사하게 차원을 맞교환할 때 사용

코드 10-64 각 단어에 대한 벡터 형태 확인

```
token_vecs_cat = [] # 형태가 [33x(33x768)]인 벡터를 [33x25,344]로 변경하여 저장
for token in token_embeddings: # token_embeddings는 [33x12x768] 형태의 텐서를 갖습니다
    cat_vec = torch.cat((token[-1], token[-2], token[-3], token[-4]), dim=0)
    token_vecs_cat.append(cat_vec)
print ('형태는: %d x %d' % (len(token_vecs_cat), len(token_vecs_cat[0])))
```

개별적 토큰은 [12x768]의 형태를 가지며, 네 개의 계층을 이어 붙이면 각 계층의 벡터는768개의 값을 갖게 되므로 'cat_vec'의 길이는 3072가 됨

이 때 'cat_vec'는 토큰을 의미함

코드 10-66 문장 벡터

```
token_vecs = hidden_states[-2][0]
sentence_embedding = torch.mean(token_vecs, dim=0)
print ("최종 임베딩 벡터의 형태:", sentence_embedding.size())
```

은닉 상태(`hidden_states`)의 형태는 [13x1×33×768]

'token_vecs'의 텐서 형태는 [33x768]

코드 10-69 코사인 유사도 계산

```
from scipy.spatial.distance import cosine
diff_apple = 1 - cosine(token_vecs_sum[5], token_vecs_sum[27]) # '사과와 많았다'와 '나에게 사과했다'에서 단어 '사과' 사이의 코사인 유사도를 계산
same_apple = 1 - cosine(token_vecs_sum[5], token_vecs_sum[16]) # '사과가 많았다', '사과를 먹었다'에 있는 '사과' 사이의 코사인 유사도를 계산
print('*유사한* 의미에 대한 벡터 유사성:  %.2f' % same_apple)
print('*다른* 의미에 대한 벡터 유사성:  %.2f' % diff_apple)
```

유사한 의미에 대한 벡터 유사성: 0.86

다른 의미에 대한 벡터 유사성: 0.91

문장에서 사용되는 '사과'라는 단어의 코사인 유사도를 측정한 결과

다국어 버트 모델을 사용하더라도 한국어에 대해서는 정확한 판별이 어려운 것을 확인할 수 있음
또한, 사과라는 단어가 한 번 더 쪼개져 있기 때문에 정확한 결과라고 하기도 어려움
한국어에 대한 임베딩은 국내에서 개발된 모델을 이용하는 것이 더 정확도가 높음

자연어 처리를 위한 임베딩 방법만 알고 있다면 언어와 상관없이 단어/문장에 대한 임베딩을 진행하며, 모델을 생성하고 훈련시킨 후 예측 및 분류를 수행할 수 있음