순환신경망 RNN

7장 순환신경망

2020.08.07금 2h

7장 순환 신경망

4 임베딩 layers

2h

임베딩 레이어(Embedding Layer)

- **임베딩 레이어**
  - 자연어를 수치화된 정보로 바꾸기 위한 레이어
    - **자연어는 시간의 흐름에 따라 정보가 연속적으로 이어지는 시퀀스 데이터**
    - **영어는 문자(character), 한글은 문자를 넘어 자소 단위로도 쪼갤 수 있음**
  - n-gram 기법
    - **몇 개의 문자를 묶어서 파악하는 방법**
- **정수 인덱스(index)를 저장하는 방법**
  - "This is a big cat"이라는 문장에 대해 정수 인덱스를 저장하면 처음 나오는 단어부터 인덱스를 저장

단어	인덱스
this	0
is	1
a	2
big	3
cat	4

  - "This is big"
    - **[0, 1, 3] 으로 표현**
    - **[0, 1, 3] 을 다시 원핫인코딩으로 표현하는 방법**

워드 임베딩

- **Word를 R차원의 Vector로 매핑시켜주는 것**
  - 값, 텍스트 내의 단어들을 밀집 벡터(dense vector)로 만드는 것
  - 단어를 의미론적 기하 공간에 매칭 시킬 수 있도록 수치 및 벡터화시키는 것
  - 밀집 벡터는 대부분의 값이 실수이고, 저차원적인 벡터를 의미
  - W는 Learning을 통해 학습

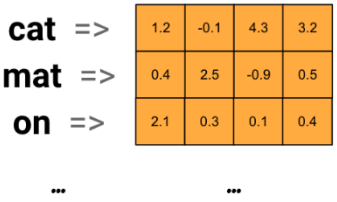
$$W(\text{“cat”}) = (0.2,\ -0.4,\ 0.7,\ \ldots)$$

$$W(\text{“mat”}) = (0.0,\ 0.6,\ -0.1,\ \ldots)$$

- **2차원 임베딩을 하는 경우 다음과 같은 숫자 벡터**

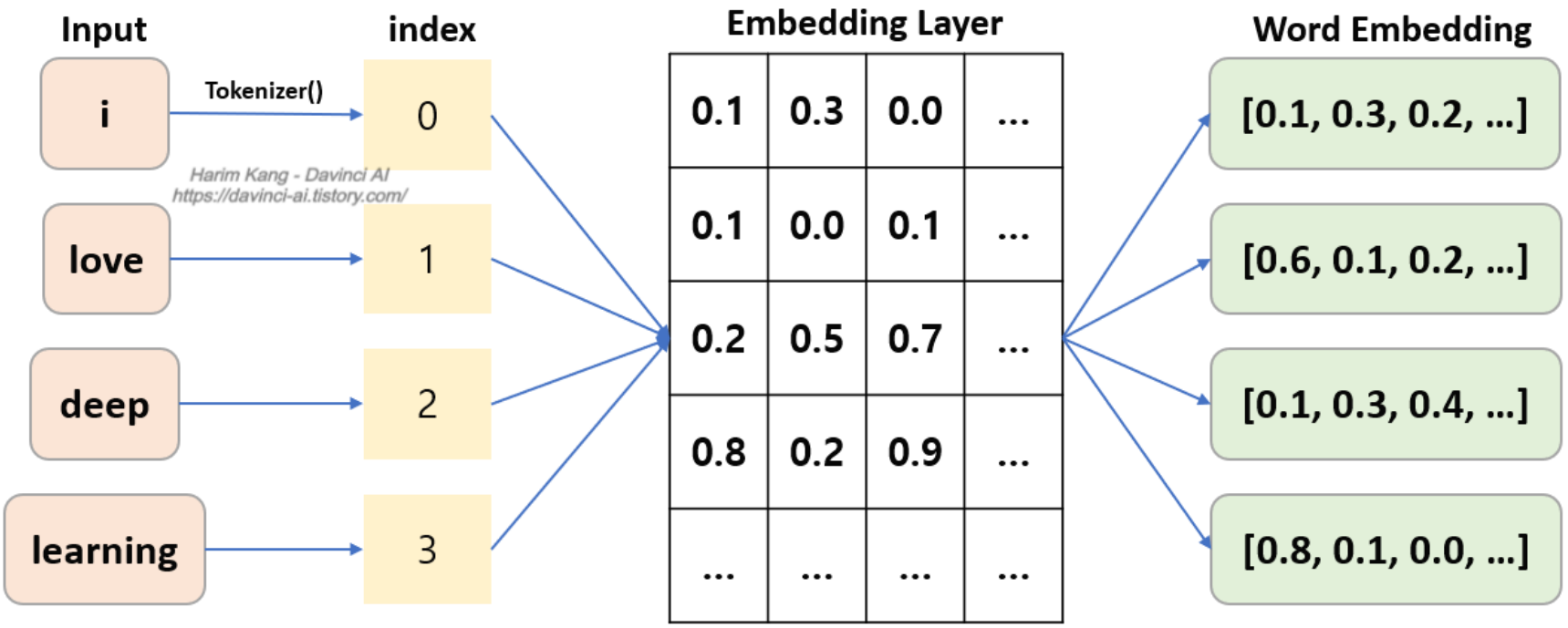
```
"I": (0.3, 0.2)
"am": (0.1, 0.8)
"a": (0.5, 0.6)
"boy": (0.2, 0.9)
"girl": (0.4, 0.7)
```

  - 4차원의 임베딩



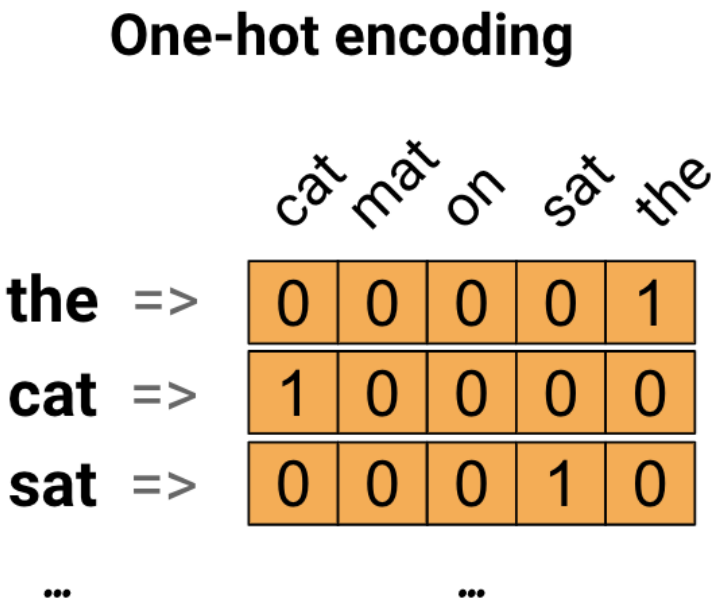
- **특징**
  - 한정된 길이의 벡터로 자연어를 표현
  - 원-핫 벡터와 같은 희소 벡터(0 또는 1로 이루어진 벡터)와 달리 훈련 데이터로부터 학습을 하는 벡터

워드 임베딩 과정


Input
index
Embedding Layer
Word Embedding
i
Tokenizer()
0
love
1
deep
2
learning
3
Harim Kang - Davinci AI
https://davinci-ai.tistory.com/
0.1 0.3 0.0 ...
0.1 0.0 0.1 ...
0.2 0.5 0.7 ...
0.8 0.2 0.9 ...
... ...
[0.1, 0.3, 0.2, ...]
[0.6, 0.1, 0.2, ...]
[0.1, 0.3, 0.4, ...]
[0.8, 0.1, 0.0, ...]

One-hot encodings

- 단어의 인덱스에 해당하는 원소만 1이고 나머지는 0인 배열
  - 약점
    - 사용하는 메모리의 양에 비해 너무 적은 정보량을 표현
    - 저장된 단어의 수가 많아질수록 원-핫 인코딩 배열의 두 번째 차원의 크기도 그에 비례해서 늘어나기 때문에 이 데이터가 차지하는 메모리의 양이 더욱 늘어남

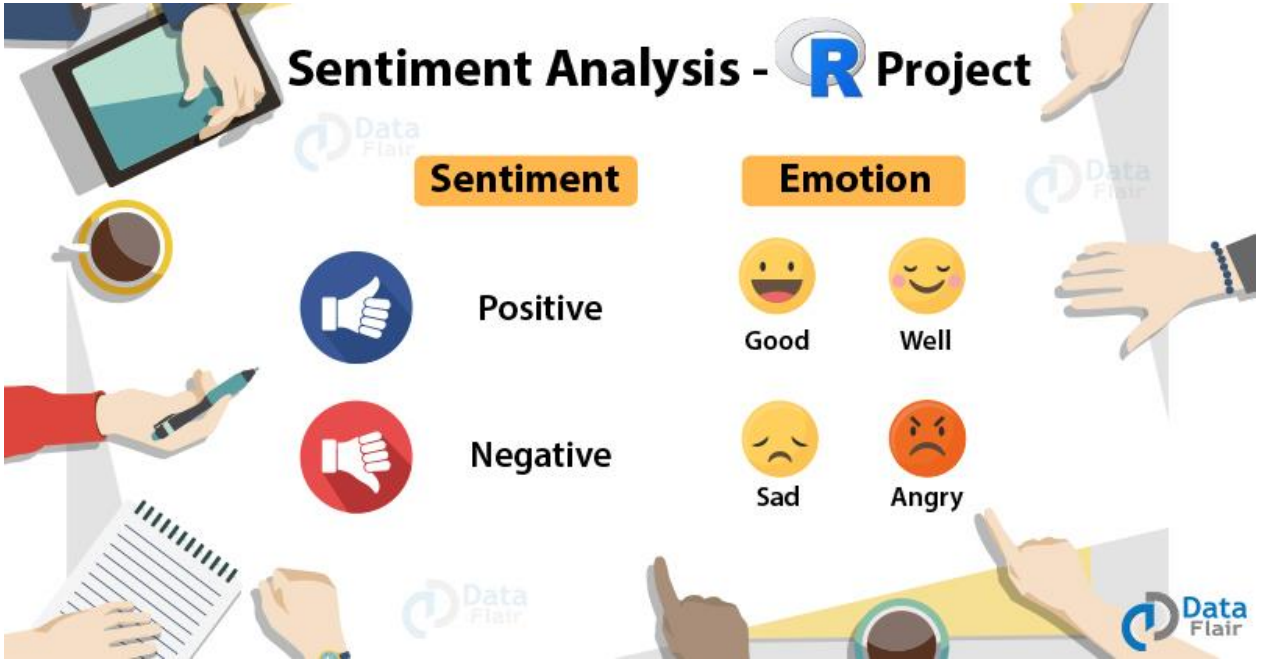


파일

- **Chapter7_study.ipynb**

감성 분석(Sentiment analysis)

- **입력된 자연어 안의 주관적 의견, 감정 등을 찾아내는 문제**
  - 문장의 긍정/부정이나 긍정/중립/부정을 분류
  - 영화 리뷰나 음식점 리뷰
    - **데이터의 양이 많고 별점을 함께 달기 때문에**
      - **긍정/중립/부정 라벨링이 쉬워서 극성(polarity) 감성 분석에 쉽게 적용이 가능**



긍정 부정 감성 분석

- **"Naver Sentiment Movie Corpus v1.0"**
  - 네이버의 박은정 박사
  - 2015년에 발표한 "Naver Sentiment Movie Corpus v1.0"을 이용
    - **긍정/부정 감성 분석**
- **20만 개 영화 리뷰**
  - 훈련 데이터 15만개, 테스트 데이터 5만개
  - 부정적인 리뷰 10만개
    - **10만 개 별점 1-4**
  - 긍정적인 리뷰 10만개
    - **10만개, 9-10**
  - 다음은 데이터에서 제외
    - **중립적 리뷰**
      - **데이터 세트에서는 제외, 별점 5-8에 해당하는 리뷰**

```
1 # 7.19 Naver Sentiment Movie Corpus v1.0 다운로드
2 path_to_train_file = tf.keras.utils.get_file('train.txt', 'https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt')
3 path_to_test_file = tf.keras.utils.get_file('test.txt', 'https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt')
```

```
Downloading data from https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt
14630912/14628807 [==============================] - 1s 0us/step
Downloading data from https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt
4898816/4893335 [==============================] - 0s 0us/step
```

데이터 로드

- **하나의 리뷰는 행으로 구분**
  - 한 행은 2 개의 탭(₩t)으로 id, 리뷰 내용, 레이블 3개로 구분
  - Label
    - **0: 부정, 1: 긍정**

```
1 # 7.20 데이터 로드 및 확인
2 # 데이터를 메모리에 불러옵니다. encoding 형식으로 utf-8 을 지정해야합니다.
3 train_text = open(path_to_train_file, 'rb').read().decode(encoding='utf-8')
4 test_text = open(path_to_test_file, 'rb').read().decode(encoding='utf-8')
5
6 # 텍스트가 총 몇 자인지 확인합니다.
7 print('Length of text: {} characters'.format(len(train_text)))
8 print('Length of text: {} characters'.format(len(test_text)))
9 print()
10
11 # 처음 300 자를 확인해봅니다.
12 print(train_text[:300])
```

```
Length of text: 6937271 characters
Length of text: 2318260 characters

id	document	label
9976970 아 더빙.. 진짜 짜증나네요 목소리	0
3819312 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
10265843	너무재밓었다그래서보는것을추천한다	0
9045019 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
6483659 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 던스트가 너무나도 이뻐보였다 1
5403919 막 걸음마 뗀 3세부터 초등학교 1학년생인 8살용영화.ㅋㅋㅋ...별반개도 아까움.	0
7797314 원작의
```

데이터(학습 및 테스트) 정답 추출

- **train_Y, test_Y를 구하는 방법**
  - 하나하나의 리뷰 분리
    - **먼저 각 텍스트를 개행 문자(₩n)로 분리한 다음**
  - 헤더에 해당하는 부분(id document label)을 제외한
    - **나머지([1:])에 대해 각 행을 처리**
      - **for row in train_text.split('₩n')[1:]**
  - 각 행은 탭 문자(₩t)로 나눠진 후에
    - **정답인 3번째 원소(원래는 정수 형태의 문자열)를 정수(integer)로 변환해서 저장**
      - **[int(row.split(' ₩t ')[2])]**
  - np.array로 결과 리스트를 감싸서 네트워크에 입력하기 쉽게 생성

```
[8]  1 # 7.21 학습을 위한 정답 데이터(Y) 만들기
     2 train_Y = np.array([[int(row.split('₩t')[2])] for row in train_text.split('₩n')[1:] if row.count('₩t') > 0])
     3 test_Y = np.array([[int(row.split('₩t')[2])] for row in test_text.split('₩n')[1:] if row.count('₩t') > 0])
     4 print(train_Y.shape, test_Y.shape)
     5 print(train_Y[:5])
```

```
(150000, 1) (50000, 1)
[[0]
 [1]
 [0]
 [0]
 [1]]
```

토큰화(Tokenization)와 정제(Cleansing)

- **토큰화**
  - 자연어를 처리 가능한 작은 단위로 나누는 것
    - **띄어쓰기 단위로 나누기**
- **정제**
  - 원하지 않는 입력이나 불필요한 기호 등을 제거하는 것
  - 정제를 위한 함수
    - **김윤 박사의 CNN_sentence 깃허브 저장소의 코드를 사용**
      - https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
    - **함수 clean_str(string)**
      - **먼저 제거할 문자는 모두 공백으로 대체하고**
      - **나눌 단어는 중간에 공백을 삽입하여**
      - **공백으로 단어를 나눌 전 처리 과정을 진행하는 함수**

clean_str(string) 함수

- re.sub(r"[^가-힣A-Za-z0-9(),!?₩'₩`]", " ", string)
  - 대괄호([])로 묶은 부분의 처음에 ^
    - **그에 포함되지 않는 나머지 모두를 선택**
  - 한글, 영문, 숫자, 괄호, 쉼표, 느낌표, 물음표, 작은따옴표('), 역따옴표(`)를 제외한 나머지는 모두 찾아서 공백(" ")으로 대체
    - **마침표 .도 공백으로 바뀌어 최종적으로 사라짐**
- **대부분은 다음과 같이 단어에 붙어있는 불피요한 부분에 공백을 추가**
  - string = re.sub(r"₩'s", " ₩'s", string)
    - **세 번째 인수인 string에서 첫 번째 인수에 해당하는 내용을 찾아서 두 번째 인수로 단순히 교체**
      - **'s를 앞에 공백을 붙이기**

```
# 7.22 train 데이터의 입력(X)에 대한 정제(Cleaning)
import re
# From https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
def clean_str(string):
    string = re.sub(r"[^가-힣A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    …

    return string.lower()
```

리뷰 내용 정제

- **train_text_X: 리뷰 내용 정제해 저장**

```
# 7.22 train 데이터의 입력(X)에 대한 정제(Cleaning)
import re
# From https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
def clean_str(string):
    string = re.sub(r"[^가-힣A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"\'s", " \'s", string)
    string = re.sub(r"\'ve", " \'ve", string)
    …

    return string.lower()
```

  - 각 행에서 탭으로 분리한 것 중 첨자 1번이 리뷰 내용

```
train_text_X = [row.split('\t')[1] for row in train_text.split('\n')[1:] if row.count
('\t') > 0]
```

  - 각각의 리뷰가 저장된 sentense를 정제하기 위해 함수 호출

```
train_text_X = [clean_str(sentence) for sentence in train_text_X]
```

- **첫 5개 리뷰의 단어 추출**

```
# 문장을 띄어쓰기 단위로 단어 분리
sentences = [sentence.split(' ') for sentence in train_text_X]
for i in range(5):
    print(sentences[i])
```

```
['아', '더빙', '진짜', '짜증나네요', '목소리']
['흠', '포스터보고', '초딩영화줄', '오버연기조차', '가볍지', '않구나']
['너무재밓었다그래서보는것을추천한다']
['교도소', '이야기구먼', '솔직히', '재미는', '없다', '평점', '조정']
['사이몬페그의', '익살스런', '연기가', '돋보였던', '영화', '!', '스파이더맨에서',
```

리뷰를 단어로만 구분하여 생성

- **정규 표현식(re:regular expression) 불러오기**

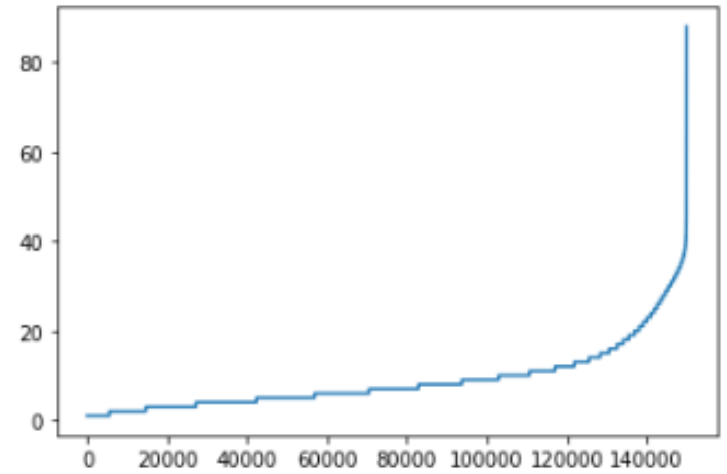
```
[9]  1 # 7.22 train 데이터의 입력(X)에 대한 정제(Cleaning)
     2 import re
     3 # From https://github.com/yoonkim/CNN_sentence/blob/master/process_data.py
     4 def clean_str(string):
     5     string = re.sub(r"[^가-힣A-Za-z0-9(),!?₩'₩`]", " ", string)
     6     string = re.sub(r"₩'s", " ₩'s", string)
     7     string = re.sub(r"₩'ve", " ₩'ve", string)
     8     string = re.sub(r"n₩'t", " n₩'t", string)
     9     string = re.sub(r"₩'re", " ₩'re", string)
    10     string = re.sub(r"₩'d", " ₩'d", string)
    11     string = re.sub(r"₩'ll", " ₩'ll", string)
    12     string = re.sub(r",", " , ", string)
    13     string = re.sub(r"!", " ! ", string)
    14     string = re.sub(r"₩(", " ₩( ", string)
    15     string = re.sub(r"₩)", " ₩) ", string)
    16     string = re.sub(r"₩?", " ₩? ", string)
    17     string = re.sub(r"₩s{2,}", " ", string)
    18     string = re.sub(r"₩'{2,}", "₩'", string)
    19     string = re.sub(r"₩'", "", string)
    20
    21     return string.lower()
    22
    23
    24 train_text_X = [row.split('₩t')[1] for row in train_text.split('₩n')[1:] if row.count('₩t') > 0]
    25 train_text_X = [clean_str(sentence) for sentence in train_text_X]
    26 # 문장을 띄어쓰기 단위로 단어 분리
    27 sentences = [sentence.split(' ') for sentence in train_text_X]
    28 for i in range(5):
    29     print(sentences[i])
```

```
['아', '더빙', '진짜', '짜증나네요', '목소리']
['흠', '포스터보고', '초딩영화줄', '오버연기조차', '가볍지', '않구나']
['너무재밋었다그래서보는것을추천한다']
['교도소', '이야기구먼', '솔직히', '재미는', '없다', '평점', '조정']
['사이몬페그의', '익살스런', '연기가', '돋보였던', '영화', '!', '스파이더맨에서', '늙어보이기만', '했던',
```

문장 길이 확인

- **네트워크에 입력하려면 데이터의 크기(문장의 단어 수)는 동일**
  - 긴 문장은 줄이고, 짧은 문장에는 공백을 의미하는 (padding)을 채워 넣어야 함
- **문장의 단어 수 그리기**
  - Y축
    - **문장의 단어 개수**
  - 15만 개의 문장 중에서 대부분이 40단어 이하로 구성되어 있음을 확인
- **특히 25 단어 이하인 리뷰의 수**
  - 142,587개로 전체의 95% 정도

```
1 # 7.23 각 문장의 단어 길이 확인
2 import matplotlib.pyplot as plt
3 sentence_len = [len(sentence) for sentence in sentences]
4 sentence_len.sort()
5 plt.plot(sentence_len)
6 plt.show()
7
8 print(sum([int(l<=25) for l in sentence_len]))
```



142587

한 리뷰인 문장의 단어를 25개로 동일하게

- **기준이 되는 문장의 길이를 25단어**
  - 이 이상은 생략
  - 이하는 패딩으로 길이를 25로 맞추어 임베딩 레이어에 넣을 준비
- **각 단어의 최대 길이도 5로 조정**
  - 5로 조정, 이후는 제거
    - **'스파이더맨에서'는 '스파이더맨' 까지**

```
1 # 7.24 단어 정제 및 문장 길이 줄임
2 sentences_new = []
3 for sentence in sentences:
4     sentences_new.append([word[:5] for word in sentence[:25])
5 sentences = sentences_new
6 for i in range(5):
7     print(sentences[i])
```

```
['아', '더빙', '진짜', '짜증나네요', '목소리']
['흠', '포스터보고', '초딩영화줄', '오버연기조', '가볍지', '않구나']
['너무재밓었']
['교도소', '이야기구먼', '솔직히', '재미는', '없다', '평점', '조정']
['사이몬페그', '익살스런', '연기가', '돋보였던', '영화', '!', '스파이더맨', '늙어보이기', '했던', '커스틴', '던스트가', '너무나도', '이뻐보였다']
```

짧은 문장도 같은 길이의 문장(25단어)으로 바꾸기

- **Tokenizer 사용**
  - 모든 단어를 사용하지 않고 출현 빈도가 가장 높은 일부 단어만 사용
  - 데이터에 출현하는 모든 단어의 개수를 세고 빈도 수로 정렬
    - **num_words**
      - **지정된 만큼만 숫자로 반환하고 나머지는 0으로 반환**
  - tokenizer.fit_on_texts(sentences)
    - **Tokenizer에 데이터를 실제로 입력**
  - tokenizer.texts_to_sequences(sentence)
    - **문장을 입력 받아 숫자를 반환**
    - **결과를 학습 데이터로 사용**
  - pad_sequences(train_X, padding='post')
    - **단어 수 25개 미만인 데이터의 끝에 0을 추가**
    - **인수 pre**
      - **문장의 앞에 패딩을 넣고**
    - **post**
      - **문장의 뒤에 패딩을 추가**

```
# 7.25 Tokenizer와 pad_sequences를 사용한 문장 전처리
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=20000)
tokenizer.fit_on_texts(sentences)
train_X = tokenizer.texts_to_sequences(sentences)
train_X = pad_sequences(train_X, padding='post')

print(train_X[:5])
```

```
[[  25  884    8 5795 1111    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0
    0]
 [ 588 5796 6697    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]
...
]
```

상위 2만개에 들지 않은 단어라 모두 0

```
['아', '더빙', '진짜', '짜증나네요', '목소리']
['흠', '포스터보고', '초딩영화줄', '오버연기조', '가볍지', '않구나']
['너무재밋었']
['교도소', '이야기구먼', '솔직히', '재미는', '없다', '평점', '조정']
['사이몬페그', '익살스런', '연기가', '돋보였던', '영화', '!', '스파이더맨',
```

tokenizer.index_word()

- **메소드 index_word(index)**
  - 해당 첨자의 단어를 반환
  - 각 번호마다 매칭되는 한글을 보려면 위와 같은 코드로 구현

```
[14]  1 for idx, word in enumerate(range(1,26), 1):
      2   print(idx, tokenizer.index_word[word])
```

```
1
2 !
3 ,
4 영화
5 ㅠ?
6 너무
7 정말
8 진짜
9 이
10 그냥
11 왜
12 이런
13 더
14 수
15 영화를
16 다
17 잘
18 보고
19 좀
20 영화는
21 영화가
22 그
23 봤는데
24 본
25 아
```

```
1 # 7.26 Tokenizer의 동작 확인
2 print(tokenizer.index_word[19999])
3 print(tokenizer.index_word[20000])
4 temp = tokenizer.texts_to_sequences(['#$#$#', '경우는', '잊혀질', '연기가'])
5 print(temp)
6 temp = pad_sequences(temp, padding='post')
7 print(temp)
```

```
경우는
잊혀질
[[], [19999], [], [106]]
[[    0]
 [19999]
 [    0]
 [  106]]
```

잊혀질은 20000번이어서 실제로는 padding에 의해 0으로 삽입

20000번 이상은 공백으로 반환

tf.keras.preprocessing.sequence.pad_sequences

```
[21]  1 sequence = [[1], [2, 3], [4, 5, 6]]
      2 tf.keras.preprocessing.sequence.pad_sequences(sequence)
```

```
array([[0, 0, 1],
       [0, 2, 3],
       [4, 5, 6]], dtype=int32)
```

```
[22]  1 tf.keras.preprocessing.sequence.pad_sequences(sequence, value=-1)
```

```
array([[-1, -1,  1],
       [-1,  2,  3],
       [ 4,  5,  6]], dtype=int32)
```

```
[23]  1 tf.keras.preprocessing.sequence.pad_sequences(sequence, padding='post')
```

```
array([[1, 0, 0],
       [2, 3, 0],
       [4, 5, 6]], dtype=int32)
```

영화 리뷰의 긍정/부정을 분류하는 네트워크 정의

- **긍정/부정을 분류하는 네트워크를 정의**
  - 임베딩 레이어와 LSTM 레이어를 연결한 뒤
  - 마지막에 Dense 레이어의 소프트맥스 활성화 함수를 사용

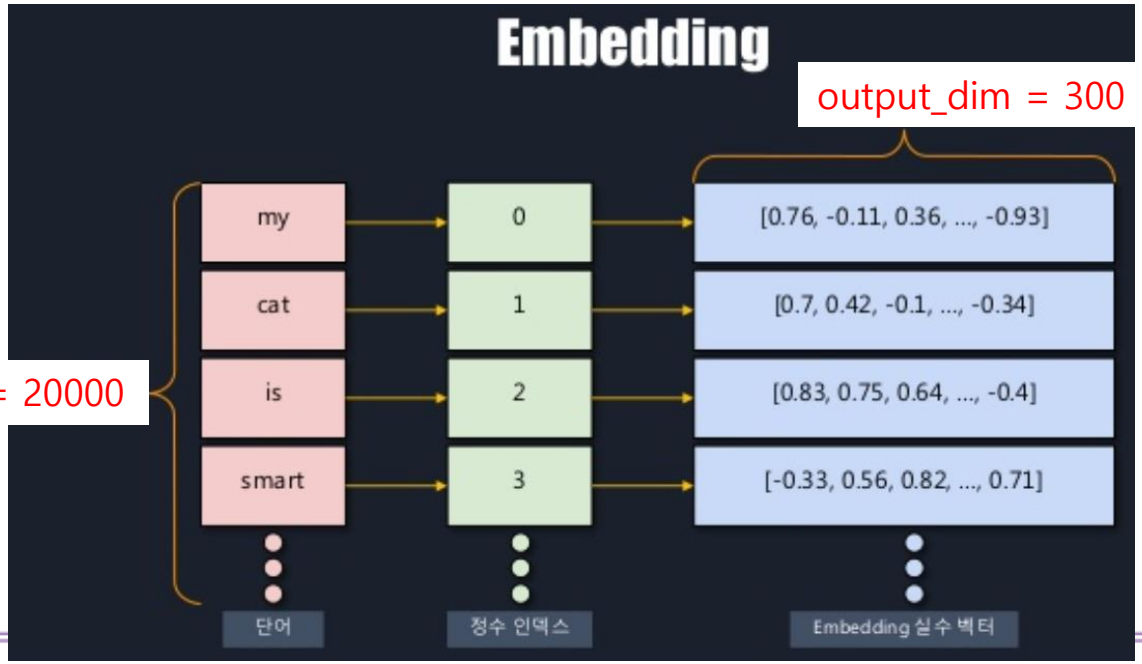
```
1 # 7.27 감성 분석을 위한 모델 정의
2 model = tf.keras.Sequential([
3     tf.keras.layers.Embedding(20000, 300, input_length=25),
4     tf.keras.layers.LSTM(units=50),
5     tf.keras.layers.Dense(2, activation='softmax')
6 ])
7
8 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
9 model.summary()
```

input_length 인수가 중요, 데이터 전처리를 25 기준으로 정해두었기 때문에, input_length로 정의

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 25, 300)           6000000
_________________________________________________________________
lstm (LSTM)                  (None, 50)                70200
_________________________________________________________________
dense (Dense)                (None, 2)                 102
=================================================================
Total params: 6,070,302
Trainable params: 6,070,302
Non-trainable params: 0
_________________________________________________________________
```

임베딩의 주요 인수

- **tf.keras.layers.Embedding(20000, 300, input_length=25)**
  - 20000 (input_dim, output_dim)
    - **2만 개의 단어만 활용**
  - 300
    - **각 문장에 들어 있는 25개의 단어를 길이 300의 임베딩 벡터로 변환**
  - input_length=25
    - **문장의 리뷰는 최대 25개의 단어**



학습

```
# 7.28 감성 분석 모델 학습
history = model.fit(train_X, train_Y, epochs=5, batch_size=128, validation_split=0.2)
```

학습 결과 시각화

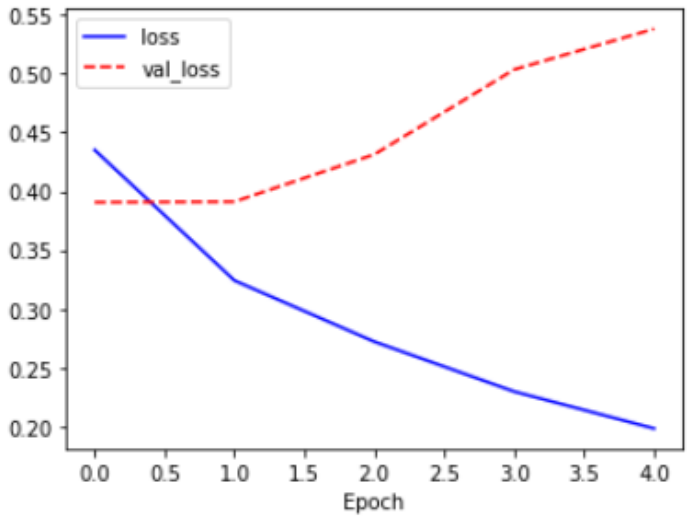
- **batch_size=128**
- **5 에포크 학습**
  - 학습 과정에서 loss는 꾸준히 감소
- **과적합 발생**
  - val_loss는 점점 증가하는 것을 확인

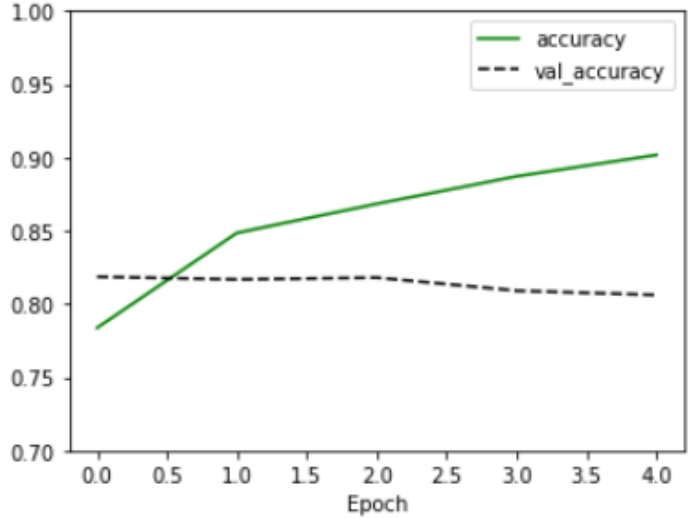
```
# 7.29 감성 분석 모델 학습 결과 확인
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], 'b-', label='loss')
plt.plot(history.history['val_loss'], 'r--
', label='val_loss')
plt.xlabel('Epoch')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], 'g-
', label='accuracy')
plt.plot(history.history['val_accuracy'], 'k--
', label='val_accuracy')
plt.xlabel('Epoch')
plt.ylim(0.7, 1)
plt.legend()

plt.show()
```





학습 결과 테스트

- **test_text에도 train_text와 같은 변환 과정을 거쳐서 test_X를 생성**
  - 이미 학습데이터로 학습시킨 Tokenizer를 어떤 변경 없이 그대로 사용
    - **테스트 데이터는 우리 손에 없다는 가정하에 작업을 진행**
- **약 80% 정도의 결과**

```
# 7.30 테스트 데이터 평가
test_text_X = [row.split('₩t')[1] for row in test_text.split('₩n')[1:] if row.count('₩t') > 0]
test_text_X = [clean_str(sentence) for sentence in test_text_X]
sentences = [sentence.split(' ') for sentence in test_text_X]
sentences_new = []
for sentence in sentences:
    sentences_new.append([word[:5] for word in sentence][:25])
sentences = sentences_new

test_X = tokenizer.texts_to_sequences(sentences)
test_X = pad_sequences(test_X, padding='post')

model.evaluate(test_X, test_Y, verbose=0)
```

```
[0.6069439649581909, 0.7994800209999084]
```

임의의 문장에 대한 감성 분석

- **순환 신경망이 입력의 변화에 따라 값이 변한다는 것을 확인**
  - 하나의 문장을 잘라서 앞에서부터 차례로 입력
  - '너무'라는 단어 '졸리고'가 나왔을 때 99%의 확률로 부정적 감성을 예측

```
# 7.31 임의의 문장 감성 분석 결과 확인
test_sentence = '재미있을 줄 알았는데 완전 실망했다. 너무 졸리고 돈이 아까웠다.'
test_sentence = test_sentence.split(' ')
test_sentences = []
now_sentence = []
for word in test_sentence:
    now_sentence.append(word)
    test_sentences.append(now_sentence[:])

test_X_1 = tokenizer.texts_to_sequences(test_sentences)
test_X_1 = pad_sequences(test_X_1, padding='post', maxlen=25)
prediction = model.predict(test_X_1)
for idx, sentence in enumerate(test_sentences):
    print(sentence)
    print(prediction[idx])
```

```
['재미있을']
[0.41499388 0.5850061 ]
['재미있을', '줄']
[0.46383956 0.53616047]
['재미있을', '줄', '알았는데']
[0.5084595 0.4915405]
['재미있을', '줄', '알았는데', '완전']
[0.52025294 0.4797471 ]
['재미있을', '줄', '알았는데', '완전', '실망했다.']
[0.52025294 0.4797471 ]
['재미있을', '줄', '알았는데', '완전', '실망했다.', '너무']
[0.59737766 0.40262237]
['재미있을', '줄', '알았는데', '완전', '실망했다.', '너무', '졸리고']
[0.99524826 0.00475175]
['재미있을', '줄', '알았는데', '완전', '실망했다.', '너무', '졸리고', '돈이']
[0.9979342  0.00206574]
['재미있을', '줄', '알았는데', '완전', '실망했다.', '너무', '졸리고', '돈이', '아까웠다.']
[0.9979342  0.00206574]
```