```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
```

```
sentences = ['nice great best amazing', 'stop lies', 'pitiful nerd', 'excellent
y_train = [1, 0, 0, 1, 1, 0, 1]
```

```
t = Tokenizer()
t.fit_on_texts(sentences)
vocab_size = len(t.word_index) + 1

print(vocab_size)
```

```
16
```

```
x_encoded = t.texts_to_sequences(sentences)
print(x_encoded)
```

```
[[1, 2, 3, 4], [5, 6], [7, 8], [9, 10], [11, 12], [13], [14, 15]]
```

```
max_len = max(len(I) for I in x_encoded)
print(max_len)
```

```
4
```

```
x_train = pad_sequences(x_encoded, maxlen=max_len, padding='post')
y_train = np.array(y_train)
print(x_train)
```

```
[[ 1  2  3  4]
 [ 5  6  0  0]
 [ 7  8  0  0]
 [ 9 10  0  0]
 [11 12  0  0]
 [13  0  0  0]
 [14 15  0  0]]
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, Flatten

model = Sequential()
model.add(Embedding(vocab_size, 4, input_length=max_len))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.fit(x_train, y_train, epochs= 100, verbose=1) # 0 1 2
```

```
Epoch 1/100
1/1 [==============================] - 1s 897ms/step - loss: 0.7003 - acc: 0.2
```

```
Epoch 2/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6987 - acc: 0.28
Epoch 3/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6971 - acc: 0.28
Epoch 4/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6956 - acc: 0.28
Epoch 5/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6940 - acc: 0.57
Epoch 6/100
1/1 [==============================] - 0s 9ms/step - loss: 0.6925 - acc: 0.571
Epoch 7/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6909 - acc: 0.71
Epoch 8/100
1/1 [==============================] - 0s 9ms/step - loss: 0.6894 - acc: 0.714
Epoch 9/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6878 - acc: 0.71
Epoch 10/100
1/1 [==============================] - 0s 8ms/step - loss: 0.6863 - acc: 0.714
Epoch 11/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6847 - acc: 0.71
Epoch 12/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6832 - acc: 0.71
Epoch 13/100
1/1 [==============================] - 0s 14ms/step - loss: 0.6817 - acc: 0.85
Epoch 14/100
1/1 [==============================] - 0s 10ms/step - loss: 0.6801 - acc: 0.85
Epoch 15/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6786 - acc: 0.85
Epoch 16/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6770 - acc: 0.85
Epoch 17/100
1/1 [==============================] - 0s 12ms/step - loss: 0.6755 - acc: 0.85
Epoch 18/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6739 - acc: 0.85
Epoch 19/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6723 - acc: 0.85
Epoch 20/100
1/1 [==============================] - 0s 12ms/step - loss: 0.6708 - acc: 1.00
Epoch 21/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6692 - acc: 1.00
Epoch 22/100
1/1 [==============================] - 0s 24ms/step - loss: 0.6676 - acc: 1.00
Epoch 23/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6660 - acc: 1.00
Epoch 24/100
1/1 [==============================] - 0s 20ms/step - loss: 0.6644 - acc: 1.00
Epoch 25/100
1/1 [==============================] - 0s 11ms/step - loss: 0.6628 - acc: 1.00
Epoch 26/100
1/1 [==============================] - 0s 12ms/step - loss: 0.6612 - acc: 1.00
Epoch 27/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6595 - acc: 1.00
Epoch 28/100
1/1 [==============================] - 0s 13ms/step - loss: 0.6579 - acc: 1.00
Epoch 29/100
1/1 [==============================] - 0s 15ms/step - loss: 0.6563 - acc: 1.00
```

▾ 네이버 영화 리뷰 감성분석 (word2Vec)

```
!pip install konlpy
```

```
Requirement already satisfied: konlpy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: colorama in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: tweepy>=3.7.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: lxml>=4.1.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: beautifulsoup4==4.6.0 in /usr/local/lib/python3
Requirement already satisfied: JPype1>=0.7.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.6 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/d
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests[socks]>=2.11.1 in /usr/local/lib/pytho
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyth
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python
```

```
import pandas as pd
import urllib.request
import matplotlib.pyplot as plt
import re
from konlpy.tag import Okt
from tensorflow import keras
from tensorflow.keras.preprocessing.text import Tokenizer
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from collections import Counter
```

데이터 준비

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ra
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/master/ra
```

```
('ratings_test.txt', <http.client.HTTPMessage at 0x7f6c9e392590>)
```

```
train_data = pd.read_table('ratings_train.txt')
test_data = pd.read_table('ratings_test.txt')
```

```
train_data. head()
```

id	document	label

```
from konlpy.tag import Mecab
!git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
%cd Mecab-ko-for-Google-Colab/
!bash install_mecab-ko_on_colab190912.sh
```

```
fatal: destination path 'Mecab-ko-for-Google-Colab' already exists and is not
/content/Mecab-ko-for-Google-Colab
Installing konlpy.....
Requirement already satisfied: konlpy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: numpy>=1.6 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: lxml>=4.1.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: tweepy>=3.7.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: JPype1>=0.7.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: beautifulsoup4==4.6.0 in /usr/local/lib/python3
Requirement already satisfied: colorama in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/d
Requirement already satisfied: requests[socks]>=2.11.1 in /usr/local/lib/pytho
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/pyth
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/d
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python
Done
Installing mecab-0.996-ko-0.9.2.tar.gz.....
Downloading mecab-0.996-ko-0.9.2.tar.gz.......
from https://bitbucket.org/eunjeon/mecab-ko/downloads/mecab-0.996-ko-0.9.2.tar
--2021-12-03 05:20:51--  https://bitbucket.org/eunjeon/mecab-ko/downloads/meca
Resolving bitbucket.org (bitbucket.org)... 104.192.141.1, 2406:da00:ff00::22cd
Connecting to bitbucket.org (bitbucket.org)|104.192.141.1|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://bbuseruploads.s3.amazonaws.com/eunjeon/mecab-ko/downloads/me
--2021-12-03 05:20:51--  https://bbuseruploads.s3.amazonaws.com/eunjeon/mecab-
Resolving bbuseruploads.s3.amazonaws.com (bbuseruploads.s3.amazonaws.com)... 5
Connecting to bbuseruploads.s3.amazonaws.com (bbuseruploads.s3.amazonaws.com)|
HTTP request sent, awaiting response... 200 OK
Length: 1414979 (1.3M) [application/x-tar]
Saving to: ‘mecab-0.996-ko-0.9.2.tar.gz.1’

mecab-0.996-ko-0.9. 100%[===================>]   1.35M  7.58MB/s    in 0.2s

2021-12-03 05:20:52 (7.58 MB/s) - ‘mecab-0.996-ko-0.9.2.tar.gz.1’ saved [14149

Done
Unpacking mecab-0.996-ko-0.9.2.tar.gz.......
Done
Change Directory to mecab-0.996-ko-0.9.2.......
installing mecab-0.996-ko-0.9.2.tar.gz........
configure
make
make check
make install
ldconfig
Done
```

```
Change Directory to /content
Downloading mecab-ko-dic-2.1.1-20180720.tar.gz.......
from https://bitbucket.org/eunjeon/mecab-ko-dic/downloads/mecab-ko-dic-2.1.1-2
--2021-12-03 05:21:08--  https://bitbucket.org/eunjeon/mecab-ko-dic/downloads/
Resolving bitbucket.org (bitbucket.org)... 104.192.141.1, 2406:da00:ff00::22cc
Connecting to bitbucket.org (bitbucket.org)|104.192.141.1|:443... connected.
HTTP request sent, awaiting response... 302 Found
```

```
tokenizer= Mecab()
```

```
def tokenize_and_remove_stopwords(data, stopwords, tokenizer):
    result = []

    for sentence in data:
        curr_data = []
        curr_data = tokenizer.morphs(sentence) # 형태소기반으로한 토큰화
        curr_data = [word for word in curr_data if not word in stopwords] # 불용

        result.append(curr_data)
    return result
```

```
# https://www.ranks.nl/stopwords/korean
stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '걍', '과', '도', '를', '
```

```
def load_data(train_data, test_data, num_words= 10000):
    # num_words : 등장 빈도 순위로 몇 번째에 해당하는 단어까지 사용할 것인가?
    # 10000을 입력하면, 등장 빈도 순위가 1~10000에 해당하는 단어만 사용. --> 단어집합의 크기 10,
    train_data.drop_duplicates(subset=['document'], inplace=True)
    test_data.drop_duplicates(subset=['document'], inplace=True)

    train_data = train_data.dropna(how='any')
    test_data = test_data.dropna(how='any')

    x_train = tokenize_and_remove_stopwords(train_data['document'], stopwords, t
    x_test = tokenize_and_remove_stopwords(test_data['document'], stopwords, tok

    words = np.concatenate(x_train).tolist()
    counter = Counter(words)
    counter = counter.most_common(num_words-4)

    vocab = ['<PAD>', '<BOS>', '<UNK>', '<UNUSED>'] + [key for key, _ in counter
    word_to_index = {word:index for index, word in enumerate(vocab)}

    def wordlist_to_Indexlist(wordlist):
        return [word_to_index[word] if word in word_to_index else word_to_index[

    x_train = list(map(wordlist_to_Indexlist, x_train))
    x_test = list(map(wordlist_to_Indexlist, x_test))

    return x_train, np.array(list(train_data['label'])), x_test, np.array(list(t

x_train, y_train, x_test, y_test, word_to_index = load_data(train_data, test_dat
print(x_train[0])
```

```
[32, 74, 919, 4, 4, 39, 228, 20, 33, 748]
```

```
index_to_word = {index: word for word, index in word_to_index.items()}
```

```
def get_encoded_sentence(sentece, word_to_index): # 한 문장
    return [word_to_index['<BOS>']]+ [word_to_index[word] if word in word_to_ind
```

```
def get_encoded_sentences(sentences, word_to_index): #여러 문장
    return [get_encoded_sentence(sentence, word_to_index) for sentence in senten
```

```
def get_decoded_sentence(encoded_sentence, index_to_word):
    return ' '.join(index_to_word[index] if index in index_to_word else '<UNK>'
```

```
def get_decoded_sentences(encoded_sentences, index_to_word):
    return [get_decoded_sentence(encoded_sentence, index_to_word) for encoded_se
```

```
get_decoded_sentence(x_train[10], index_to_word)
```

```
'. 진짜 짱 다 ♥'
```

▾ 모델 구성을 위한 데이터 분석 및 가공

```
total_data_text = list(x_train) + list(x_test)
num_tokens = [len(tokens) for tokens in total_data_text]
num_tokens = np.array(num_tokens)

print('문장길이 평균: ', np.mean(num_tokens))
print('문장길이 최대: ', np.max(num_tokens))
print('문장길의 표준편차: ', np.std(num_tokens))
```

```
문장길이 평균:  15.969355837799927
문장길이 최대:  116
문장길의 표준편차:  12.843536204665021
```

```
# 최대길이 (평균 + 2 * 표준편차)
max_tokens = np.mean(num_tokens) + 2 * np.std(num_tokens)
maxlen = int(max_tokens)
print('pad sequences maxlen :', maxlen)
print('전체 문장의 {}%가 maxlen설정값 이내에 포함됩니다.'.format(np.sum(num_tokens < max_
```

```
pad sequences maxlen : 41
전체 문장의 93.42988343341575%가 maxlen설정값 이내에 포함됩니다.
```

```
x_train = keras.preprocessing.sequence.pad_sequences(x_train, value = word_to_in
x_test = keras.preprocessing.sequence.pad_sequences(x_test, value = word_to_inde
print(x_train.shape)
print(x_test.shape)
```

```
(146182, 41)
(49157, 41)
```

모델 구성 및 validation 구성

```
vocab_size = 10000
word_vector_dim = 256 # 워드 벡터의 차원 수

# 1. RNN버전

model_rnn = keras.Sequential()
model_rnn.add(keras.layers.Embedding(vocab_size, word_vector_dim, input_shape=(N
model_rnn.add(keras.layers.LSTM(16, activation='relu'))
model_rnn.add(keras.layers.Dense(16, activation='relu'))
model_rnn.add(keras.layers.Dense(1, activation='sigmoid'))


# 2. 1D-CNN

model_cnn = keras.Sequential()
model_cnn.add(keras.layers.Embedding(vocab_size, word_vector_dim, input_shape=(N
model_cnn.add(keras.layers.Conv1D(16, 3, activation='relu'))
model_cnn.add(keras.layers.MaxPool1D(2))
model_cnn.add(keras.layers.Conv1D(16, 3, activation='relu'))
model_cnn.add(keras.layers.GlobalAveragePooling1D())
model_cnn.add(keras.layers.Dense(8, activation='relu'))
model_cnn.add(keras.layers.Dense(1, activation='sigmoid'))
#각 모델을 각각 다른 변수에 저장해주세요!
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't me
```

```
model_rnn.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, None, 256)         2560000

 lstm (LSTM)                 (None, 16)                17472

 dense_1 (Dense)             (None, 16)                272

 dense_2 (Dense)             (None, 1)                 17

=================================================================
Total params: 2,577,761
Trainable params: 2,577,761
Non-trainable params: 0
_________________________________________________________________
```

```
model_cnn.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_3 (Embedding)     (None, None, 256)         2560000   
                                                                 
 conv1d (Conv1D)             (None, None, 16)          12304     
                                                                 
 max_pooling1d (MaxPooling1D  (None, None, 16)         0         
 )                                                               
                                                                 
 conv1d_1 (Conv1D)           (None, None, 16)          784       
                                                                 
 global_average_pooling1d (G  (None, 16)               0         
 lobalAveragePooling1D)                                          
                                                                 
 dense_5 (Dense)             (None, 8)                 136       
                                                                 
 dense_6 (Dense)             (None, 1)                 9         
                                                                 
=================================================================
Total params: 2,573,233
Trainable params: 2,573,233
Non-trainable params: 0
_________________________________________________________________
```

```
x_val = x_train[:50000]
y_val = y_train[:50000]

partial_x_train = x_train[50000:]
partial_y_train = y_train[50000:]
```

```
model_rnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura
```

```
epochs = 15
history_rnn = model_rnn.fit(partial_x_train, partial_y_train, epochs = epochs, b
```

```
Epoch 1/15
188/188 [==============================] - 31s 154ms/step - loss: 0.4593 - acc
Epoch 2/15
188/188 [==============================] - 29s 153ms/step - loss: 0.3270 - acc
Epoch 3/15
188/188 [==============================] - 28s 150ms/step - loss: 0.2950 - acc
Epoch 4/15
188/188 [==============================] - 28s 148ms/step - loss: 0.2707 - acc
Epoch 5/15
188/188 [==============================] - 28s 150ms/step - loss: 0.2492 - acc
Epoch 6/15
188/188 [==============================] - 28s 147ms/step - loss: 0.2288 - acc
Epoch 7/15
188/188 [==============================] - 29s 153ms/step - loss: 0.2066 - acc
Epoch 8/15
188/188 [==============================] - 28s 149ms/step - loss: 0.1879 - acc
```

```
Epoch 9/15
188/188 [==============================] - 28s 150ms/step - loss: 0.1700 - acc
Epoch 10/15
188/188 [==============================] - 28s 149ms/step - loss: 0.1577 - acc
Epoch 11/15
188/188 [==============================] - 27s 146ms/step - loss: 0.1422 - acc
Epoch 12/15
188/188 [==============================] - 28s 151ms/step - loss: 0.1278 - acc
Epoch 13/15
188/188 [==============================] - 28s 149ms/step - loss: 0.1185 - acc
Epoch 14/15
188/188 [==============================] - 28s 147ms/step - loss: 0.1083 - acc
Epoch 15/15
188/188 [==============================] - 27s 146ms/step - loss: 0.0966 - acc
```

```
# CNN1D학습
model_cnn.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accura
history_cnn = model_cnn.fit(partial_x_train, partial_y_train, epochs = epochs, b
```

```
Epoch 1/15
188/188 [==============================] - 11s 20ms/step - loss: 0.5947 - accu
Epoch 2/15
188/188 [==============================] - 3s 17ms/step - loss: 0.4801 - accur
Epoch 3/15
188/188 [==============================] - 3s 17ms/step - loss: 0.4281 - accur
Epoch 4/15
188/188 [==============================] - 3s 17ms/step - loss: 0.3880 - accur
Epoch 5/15
188/188 [==============================] - 3s 17ms/step - loss: 0.3536 - accur
Epoch 6/15
188/188 [==============================] - 3s 16ms/step - loss: 0.3217 - accur
Epoch 7/15
188/188 [==============================] - 3s 17ms/step - loss: 0.2934 - accur
Epoch 8/15
188/188 [==============================] - 3s 16ms/step - loss: 0.2685 - accur
Epoch 9/15
188/188 [==============================] - 3s 17ms/step - loss: 0.2480 - accur
Epoch 10/15
188/188 [==============================] - 3s 16ms/step - loss: 0.2304 - accur
Epoch 11/15
188/188 [==============================] - 3s 17ms/step - loss: 0.2144 - accur
Epoch 12/15
188/188 [==============================] - 3s 16ms/step - loss: 0.2020 - accur
Epoch 13/15
188/188 [==============================] - 3s 16ms/step - loss: 0.1907 - accur
Epoch 14/15
188/188 [==============================] - 3s 17ms/step - loss: 0.1817 - accur
Epoch 15/15
188/188 [==============================] - 3s 17ms/step - loss: 0.1736 - accur
```

```
result_rnn = model_rnn.evaluate(x_test, y_test, verbose=2)
result_cnn = model_cnn.evaluate(x_test, y_test, verbose=2)
```

```
1537/1537 - 15s - loss: 0.6174 - accuracy: 0.8362 - 15s/epoch - 9ms/step
1537/1537 - 4s - loss: 0.7247 - accuracy: 0.8167 - 4s/epoch - 3ms/step
```

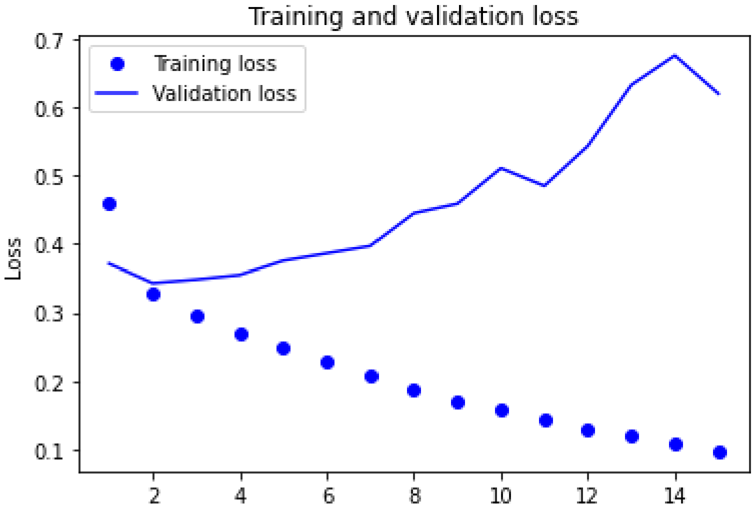
```
history_rnn_dic = history_rnn.history
```

```
history_cnn_dic = history_cnn.history
```

```
acc = history_rnn_dic['accuracy']
val_acc = history_rnn_dic['val_accuracy']
loss = history_rnn_dic['loss']
val_loss = history_rnn_dic['val_loss']

epochs = range(1, len(acc)+1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

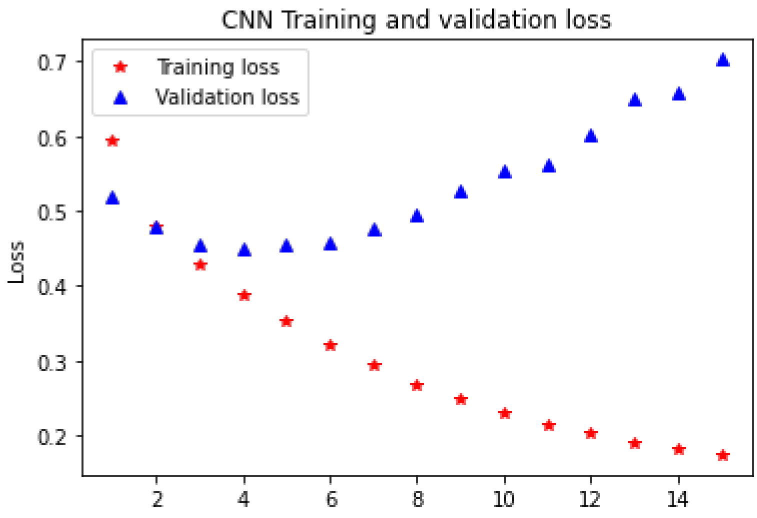


```
plt.clf()
plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

```
acc = history_cnn_dic['accuracy']
val_acc = history_cnn_dic['val_accuracy']
loss = history_cnn_dic['loss']
val_loss = history_cnn_dic['val_loss']

epochs = range(1, len(acc)+1)

plt.plot(epochs, loss, 'r*', label='Training loss')
plt.plot(epochs, val_loss, 'b^', label='Validation loss')
plt.title('CNN Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



학습된 embedding 레이어 분석

```
import os

word2vec_file_path = 'word2vec.txt'
f = open(word2vec_file_path, 'w')
f.write('{} {} \n'.format(vocab_size-4, word_vector_dim))

vectors = model_rnn.get_weights()[0]
for i in range(4, vocab_size):
  f.write('{} {}\n'.format(index_to_word[i], ' '.join(map(str, list(vectors[i, :
f.close()
```

```
from gensim.models.keyedvectors import Word2VecKeyedVectors

word_vector = Word2VecKeyedVectors.load_word2vec_format(word2vec_file_path, bina
vector = word_vector['짜증']
vector
```

```
array([-0.11695278,  0.06496852,  0.11551108,  0.12568538, -0.07970354,
       -0.04288113,  0.05794413, -0.02350133, -0.10211213, -0.09533374,
        0.13843657, -0.04619869, -0.13491851, -0.04964797, -0.05194288,
        0.06935064,  0.13430803, -0.06811351, -0.0340673 , -0.03686334,
       -0.04177307,  0.01751819, -0.0012929 ,  0.13123968, -0.02644559,
       -0.11099007, -0.08881964,  0.08301652, -0.16866721,  0.08890183,
       -0.1591071 , -0.03367618,  0.01445628, -0.14172013, -0.14004861,
        0.01477614, -0.06323729,  0.02159893, -0.2044182 ,  0.03747103,
       -0.16572665,  0.02420673, -0.0250796 ,  0.13370655,  0.1380928 ,
       -0.05256239,  0.12193506, -0.09592973,  0.11720952,  0.0650732 ,
       -0.10352246,  0.11149633,  0.16006595, -0.07323296, -0.06998923,
       -0.01254552, -0.14201644, -0.0944348 , -0.11337635, -0.08307157,
       -0.07780112,  0.06927288, -0.06150954,  0.10231654,  0.11254925,
       -0.1201505 ,  0.028915  , -0.05196309,  0.05550084,  0.05977367,
        0.17526333, -0.02987548,  0.09202926,  0.0944039 ,  0.07080351,
        0.07670016, -0.00422931, -0.0931441 ,  0.04246465, -0.00404961,
       -0.01440467, -0.05921483, -0.10580222,  0.02305742, -0.00678742,
       -0.00706935, -0.09848319,  0.0536371 , -0.09570051, -0.05298399,
        0.02704476,  0.0035957 ,  0.10261422, -0.06405214, -0.03775536,
        0.18452667,  0.03288504, -0.12324423, -0.07741722,  0.07954046,
        0.00656211, -0.05153873, -0.01645163,  0.06369604,  0.0133242 ,
        0.10092181, -0.02779633, -0.15497029, -0.01790283, -0.04249406,
        0.14783876, -0.03150017,  0.03813526, -0.01522378, -0.03545158,
        0.07818934, -0.06095735,  0.01012779,  0.05671621, -0.06144512,
       -0.0947127 , -0.11091409, -0.08188409, -0.12092128,  0.00322306,
        0.1147218 ,  0.14491603,  0.03663339, -0.05685823,  0.04844633,
       -0.07125527, -0.11948869, -0.02255244, -0.01293609,  0.00379035,
        0.07920301,  0.11697295, -0.12464049, -0.01973862,  0.02044727,
       -0.16386497,  0.02209048, -0.0030039 ,  0.05520678,  0.07522133,
        0.0792769 ,  0.0930763 , -0.06786194, -0.01087716, -0.10769738,
       -0.099434  , -0.03240903, -0.01933603, -0.08182439,  0.180865  ,
        0.09779242,  0.05587322, -0.10050135, -0.10929764,  0.05282233,
       -0.15609953, -0.06158772, -0.05481772,  0.00896752,  0.06684638,
       -0.12110747,  0.03960775, -0.06897014,  0.07884336,  0.136533  ,
        0.09475495, -0.02123453,  0.08075268,  0.0959774 , -0.03311311,
        0.10657465,  0.00506846, -0.02397223,  0.03036373, -0.04405931,
       -0.0182365 , -0.00161385,  0.00397808,  0.04978829, -0.05828398,
        0.02544523, -0.09262484, -0.09010168,  0.10823145, -0.01717615,
       -0.0663865 , -0.14877994, -0.13539647, -0.11286709, -0.0681342 ,
       -0.02743596, -0.12591423,  0.08419674,  0.08989408,  0.05661532,
        0.09314044, -0.18723023,  0.20653892, -0.06358235, -0.09634232,
       -0.04697644, -0.13988325,  0.02426997,  0.10948779, -0.02153733,
       -0.08195125, -0.01065093,  0.0628752 ,  0.11958005, -0.11493804,
       -0.08486564,  0.05075926, -0.00496746, -0.16210026,  0.03710253,
        0.03285007,  0.11750418,  0.06498042, -0.11736121,  0.10307929,
       -0.06785111,  0.0354523 ,  0.07320311,  0.11654483,  0.11596978,
        0.07431025, -0.03205619, -0.05172541, -0.06347186, -0.04685179,
        0.12250837, -0.09406844,  0.04406658, -0.11494498, -0.10696688,
       -0.11044575, -0.14652328,  0.08920863,  0.07711838,  0.04863311,
        0.0375802 , -0.02296215,  0.02604582, -0.08657025, -0.03621616,
       -0.12954336, -0.01290414, -0.0691283 , -0.03963342, -0.0071046 ,
```

```
word_vector.similar_by_word("짜증")
```

```
[('식상', 0.8202544450759888),
 ('비추', 0.8075307607650757),
 ('방해', 0.8051162958145142),
 ('한계', 0.8025670051574707),
 ('쓰레기', 0.7882331013679504),
 ('이경영', 0.787883996963501),
 ('커녕', 0.7789846658706665),
 ('심형래', 0.7746418714523315),
 ('흑역사', 0.7732547521591187),
 ('수면제', 0.7686705589294434)]
```

```
word_vector.similar_by_word("재미")
```

```
[('소속', 0.6622107625007629),
 ('되게', 0.6533287763595581),
 ('흥미', 0.5998678207397461),
 ('부턴', 0.5962098240852356),
 ('할머니', 0.5617274045944214),
 ('가지', 0.5598143339157104),
 ('토요', 0.558756947517395),
 ('이건', 0.5541054606437683),
 ('천하', 0.5445069670677185),
 ('한데', 0.5438485741615295)]
```

한국어 word2vec 임베딩을 활용해서 성능 개선

```
import gensim

word2vec_path = '/content/drive/MyDrive/Colab Notebooks/영우4기_자연어 (10일완성)/dat
word2vec = gensim.models.Word2Vec.load(word2vec_path)
vector = word2vec['감동']
vector
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:5: DeprecationWar
  """
array([-1.4411083 , -0.78142536,  2.453768  , -0.86445606,  0.24195324,
        0.36006922, -1.0639709 ,  0.85037315, -1.0184479 ,  0.21196692,
       -0.6679936 ,  0.6389819 , -0.26964295,  0.66028535,  0.39613017,
        0.6428288 ,  0.6648245 ,  0.08363932, -0.2540794 ,  0.55310047,
        0.63392764,  0.19311981, -0.4648248 ,  0.1490374 ,  2.0762694 ,
        0.7872764 , -0.21711552, -0.59049505, -1.3264338 , -0.06233318,
        1.2550159 , -0.05334642, -0.5453753 , -0.8736315 ,  0.5253877 ,
        0.03122815,  0.8280226 ,  0.23597455,  0.06136359,  1.5114233 ,
       -0.340495  ,  0.912277  ,  0.5899006 , -1.3025732 , -0.87596595,
        0.21738248,  1.0366931 , -0.55709684, -0.9039502 ,  0.28133616,
       -1.7572548 , -0.29769212, -0.14536098,  0.5850025 , -0.6111547 ,
       -0.29829553,  1.4106004 , -0.38685524,  0.4801454 ,  0.40166005,
        0.28174093,  1.6133646 , -0.8590998 ,  0.49886975,  0.38605362,
       -0.1607663 , -0.87983316,  0.21996935,  0.68561727, -0.8434425 ,
        0.02520839, -0.8017276 , -0.4882501 ,  0.5937627 , -0.22273438,
       -2.1169198 ,  0.11167947,  1.2840736 ,  0.37050653, -0.49218208,
       -0.38447312, -0.04923964,  0.818749  ,  0.14430618,  0.12984185,
        1.3372396 , -0.27832717, -0.4163464 , -1.2846806 ,  0.22243507,
        1.4398693 , -0.62261546,  0.85881597,  0.35206348, -0.7983542 ,
       -0.5648404 , -0.80835617,  1.0770288 ,  0.9198583 ,  0.24598446,
       -0.339867  , -0.20509662, -0.68669695, -0.00623814, -0.6275429 ,
        2.329253  ,  0.33655322, -0.1590611 , -0.4120386 , -1.5890588 ,
        0.37837315,  0.846773  , -0.3125741 , -0.748276  ,  1.5007688 ,
       -1.5616585 , -0.33911368, -0.9860547 ,  0.27350205, -0.17658691,
        1.2938571 , -0.04034536,  0.80943936, -2.320362  ,  0.49740836,
        0.4615926 , -0.31530836, -1.782714  ,  0.25635827,  0.02759444,
        1.0618365 ,  1.3092015 ,  1.1857753 ,  0.01812731,  0.9814533 ,
        0.14263195,  0.899134  , -0.8500094 , -1.7148823 , -0.43185592,
        0.415446  ,  1.6975207 , -0.3643097 , -0.6528986 , -0.18492346,
        0.31824046,  0.12755737,  0.92763597, -0.00657997,  0.3986357 ,
       -0.07185496,  0.26286292,  0.39870608,  0.20413135,  0.99198246,
        1.4120847 , -0.756356  ,  1.3712298 , -0.81753194, -0.44601068,
        0.05893052, -0.21582486, -0.8041302 ,  0.9012229 ,  0.02169448,
       -2.1358564 ,  0.63579637,  1.9117936 , -0.6807319 ,  1.4326098 ,
       -1.156477  , -0.41411826,  1.0657284 , -1.3112395 ,  0.4747043 ,
        0.5321504 ,  0.0543594 , -0.41400573, -0.96152973, -0.06286933,
       -0.9848911 , -0.96814924,  0.05603024, -0.3239202 ,  0.7511623 ,
        1.127266  ,  0.09363312, -0.14667176,  0.19069506,  0.23503011,
        0.42607984, -0.36619186,  0.74711823,  0.47436306, -0.6458395 ,
        0.9339805 ,  0.5576014 ,  0.41820145, -0.01330989, -0.36533296],
      dtype=float32)
```

```
word2vec.similar_by_word('재미')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWar
  """Entry point for launching an IPython kernel.
[('취미', 0.5857348442077637),
 ('유머', 0.5140613913536072),
 ('매력', 0.5105490684509277),
 ('흥미', 0.4988338351249695),
 ('공짜', 0.4960595667362213),
 ('일자리', 0.49294644594192505),
 ('즐거움', 0.48700767755508423),
 ('비애', 0.4836210310459137),
 ('관객', 0.48286449909210205),
 ('향수', 0.4823310971260071)]
```

```
word2vec.similar_by_word('로맨틱')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWar
  """Entry point for launching an IPython kernel.
[('코미디', 0.7576364278793335),
 ('로맨스', 0.7021660804748535),
 ('스릴러', 0.6693054437637329),
 ('개그', 0.6552960872650146),
 ('주제곡', 0.6495761871337891),
 ('뮤지컬', 0.6382305026054382),
 ('시트콤', 0.6167846322059631),
 ('서부극', 0.6151247620582581),
 ('연극과', 0.6083630323410034),
 ('서정적', 0.5965933799743652)]
```

```
word2vec.similar_by_word('하트')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: DeprecationWar
  """Entry point for launching an IPython kernel.
[('브렛', 0.6617691516876221),
 ('숀', 0.6434553861618042),
 ('실버', 0.6352944374084473),
 ('올드', 0.6341021060943604),
 ('랄프', 0.6211183071136475),
 ('로저스', 0.6094454526901245),
 ('키드', 0.6048450469970703),
 ('피어스', 0.6011062860488892),
 ('에드', 0.6008332967758179),
 ('바우어', 0.5989881753921509)]
```

```
mecab = Mecab()
```

```
def sentiment_predict(new_sentence):
    import re
    from tensorflow.keras.preprocessing.text import Tokenizer
    from tensorflow.keras.preprocessing.sequence import pad_sequences
    t = Tokenizer()
    new_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣]','',new_sentence)
    new_sentence = mecab.morphs(new_sentence)
    new_sentence = [word for word in new_sentence if not word in stopwords]
    encoded = t.texts_to_sequences([new_sentence])
    pad_new = pad_sequences(encoded, maxlen=max_len)
    score = float(model_rnn.predict(pad_new))

    if (score > 0.5): # 긍정
        print("{:.2f}% 확률로 긍정 리뷰 입니다. \n".format(score*100))
    else:
        print("{:.2f}% 확률로 부정 리뷰 입니다. \n".format((1-score)*100))
```

```
sentiment_predict('이 영화 꿀잼 ㅋㅋㅋㅋ짱짱짱')
```

```
58.32% 확률로 부정 리뷰 입니다.
```

```
sentiment_predict('ㅋㅋㅋㅋㅋㅋㅋ')
```

```
58.32% 확률로 부정 리뷰 입니다.
```

```
sentiment_predict('재미없다')
```

```
58.32% 확률로 부정 리뷰 입니다.
```

네이버 쇼핑 리뷰 감성 분류하기

- 총 200,000개 리뷰로 구성
- 평점이 5점 만점에 1, 2, 4, 5인 리뷰들로 구성된 데이터
- 3점인 리뷰는 긍부정 유무가 애매해서 제외
- 평점이 4, 5인 리뷰에 긍정 ---> 1
- 평점이 1, 2인 리뷰에 부정 ---> 0

```
from konlpy.tag import Mecab
!git clone https://github.com/SOMJANG/Mecab-ko-for-Google-Colab.git
%cd Mecab-ko-for-Google-Colab/
!bash install_mecab-ko_on_colab190912.sh
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import urllib.request
from collections import Counter
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
urllib.request.urlretrieve("https://raw.githubusercontent.com/bab2min/corpus/mas
```

```
('ratings_total.txt', <http.client.HTTPMessage at 0x7f6c7fee8dd0>)
```

```
!pwd
```

```
/content
```

```
cd ../
```

```
/content
```

```
total_data = pd.read_table('ratings_total.txt', names=['ratings','reviews'])
print('전체 리뷰 갯수 :', len(total_data))
```

```
전체 리뷰 갯수 : 200000
```

```
total_data[:5]
```

	ratings	reviews
0	5	배공빠르고 굿
1	2	택배가 엉망이네용 저희집 밑에층에 말도없이 놔두고가고
2	5	아주좋아요 바지 정말 좋아서2개 더 구매했어요 이가격에 대박입니다. 바느질이 조금 ...
3	2	선물용으로 빨리 받아서 전달했어야 하는 상품이었는데 머그컵만 와서 당황했습니다. 전...
4	5	민트색상 예뻐요. 옆 손잡이는 거는 용도로도 사용되네요 ㅎㅎ

▾ 훈련데이터와 테스트데이터를 분리

```
total_data['label'] = np.select([total_data.ratings >3], [1], default=0)
total_data[:5]
```

	ratings	reviews	label
0	5	배공빠르고 굿	1
1	2	택배가 엉망이네용 저희집 밑에층에 말도없이 놔두고가고	0
2	5	아주좋아요 바지 정말 좋아서2개 더 구매했어요 이가격에 대박입니다. 바느질이 조금 ...	1
3	2	선물용으로 빨리 받아서 전달했어야 하는 상품이었는데 머그컵만 와서 당황했습니다. 전...	0
4	5	민트색상 예뻐요. 옆 손잡이는 거는 용도로도 사용되네요 ㅎㅎ	1

```
total_data['ratings'].nunique()
```

```
4
```

```
total_data['reviews'].nunique() # 특이값/ 고유 값 갯수 확인
```

```
199908
```

```
total_data['label'].nunique()
```

```
2
```

```
total_data.drop_duplicates(subset=['reviews'], inplace=True) # 삭제
print('샘플의 수 :', len(total_data)) # 삭제 후 갯수 확인
```

```
샘플의 수 : 199908
```

```
print(total_data.isnull().values.any())
```

```
False
```

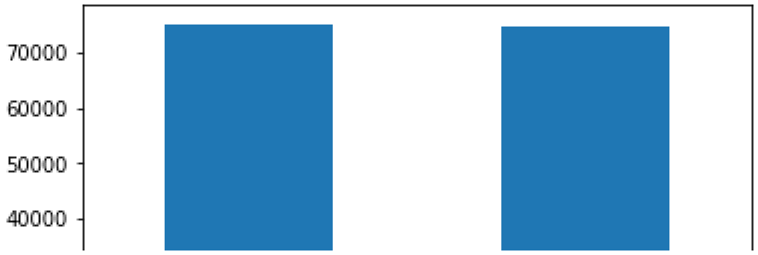
```
train_data, test_data = train_test_split(total_data, test_size=0.25, random_stat
print('훈련용 리뷰의 갯수 :', len(train_data))
print('테스트용 리뷰의 갯수 :', len(test_data))
```

```
훈련용 리뷰의 갯수 : 149931
테스트용 리뷰의 갯수 : 49977
```

레이블의 분포 확인

```
train_data['label'].value_counts().plot(kind='bar')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6c99bd57d0>
```



```
print(train_data.groupby('label').size().reset_index(name='count'))
```

```
   label  count
0      0  74918
1      1  75013
```

데이터 정제하기

```
train_data['reviews'] = train_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣]","")
```

```
train_data['reviews'].replace('', np.nan, inplace=True)
```

```
print(train_data.isnull().sum())
```

```
ratings    0
reviews    0
label      0
dtype: int64
```

```
# test data
# 중복 제거
# 정규표현식을 이용하여 한글 외 문자 제거
# 공백을 null 변경
# Null값 제거

# test_data 갯수 반환
test_data.drop_duplicates(subset=['reviews'], inplace=True)
test_data['reviews'] = test_data['reviews'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣]","")
test_data['reviews'].replace('', np.nan, inplace=True)
test_data = test_data.dropna(how='any')

print('전처리 후 테스트용 샘플의 갯수 :', len(test_data))
```

```
전처리 후 테스트용 샘플의 갯수 : 49977
```

토큰화

```
mecab= Mecab()
print(mecab.morphs('이런 상품도 상품인가요? 허허허'))
```

```
['이런', '상품', '도', '상품', '인가요', '?', '허허허']
```

불용어 제거

```
stopwords = ['의', '가', '이', '은', '들', '는', '좀', '잘', '걍', '과', '도', '를', '
```

```
train_data['tokenized'] = train_data['reviews'].apply(mecab.morphs)
train_data['tokenized'] = train_data['tokenized'].apply(lambda x: [item for item
```

```
test_data['tokenized'] = test_data['reviews'].apply(mecab.morphs)
test_data['tokenized'] = test_data['tokenized'].apply(lambda x: [item for item i
```

단어와 길이 분포 확인하기

```
negative_words = np.hstack(train_data[train_data.label==0]['tokenized'].values)
positive_words = np.hstack(train_data[train_data.label==1]['tokenized'].values)
```

```
negative_word_count = Counter(negative_words)
print(negative_word_count.most_common(20))
```

```
[('고', 38797), ('네요', 29687), ('하', 28884), ('는데', 19748), ('안', 18779), ('
```

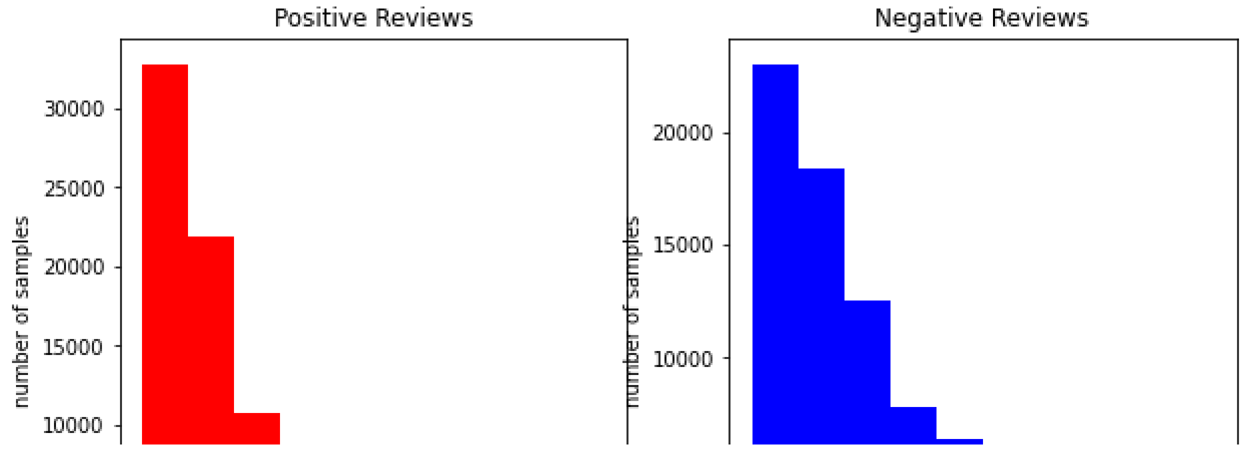
```
positive_words_count = Counter(positive_words)
print(positive_words_count.most_common(20))
```

```
[('고', 42094), ('좋', 38612), ('하', 31333), ('아요', 20203), ('네요', 18965), ('
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10,5))
text_len = train_data[train_data['label']==1]['tokenized'].map(lambda x: len(x))
ax1.hist(text_len, color='red')
ax1.set_title('Positive Reviews')
ax1.set_xlabel('length of samples')
ax1.set_ylabel('number of samples')
print('긍정 리뷰의 평균 길이 ;', np.mean(text_len))

text_len = train_data[train_data['label']==0]['tokenized'].map(lambda x: len(x))
ax2.hist(text_len, color='blue')
ax2.set_title('Negative Reviews')
ax2.set_xlabel('length of samples')
ax2.set_ylabel('number of samples')
print('부정 리뷰의 평균 길이 ;', np.mean(text_len))
```

```
긍정 리뷰의 평균 길이 ; 14.276525402263607
부정 리뷰의 평균 길이 ; 17.675525240930085
```



```
train_data.head()
```

	ratings	reviews	label	tokenized
59666	2	사이즈를센치씩늘린건데도작아요그리고색상은완전달라요칙칙한핑크네요ㅠㅠ많이아쉽지만암막효과는좋아요	0	[사이즈, 센치, 씩, 늘린, 건데, 작, 아요, 그리고, 색상, 완전, 달라요, ...
12433	2	ㅂ불만족빗이아픔멍이피부에빗질못해주겟네요	0	[ㅂ, 불, 만족, 빗이, 아, 픔멍이피부에빗질못해주겟네요]

```
test_data.head()
```

	ratings	reviews	label	tokenized
193242	1	너무낮고솜도적고실망스럽습니다	0	[너무, 낮, 고, 솜, 적, 고, 실망, 스럽, 습니다]
125080	1	피부에뾰루지가많이올라와요	0	[피부, 뾰루지, 많이, 올라, 와요]
122750	5	배송도빠르네요가격대비좋은것같아요첨에는힘들어하나조금지나니잘하	1	[배송, 빠르, 네요, 가격, 대비, 좋, 것, 같, 아요, 첨, 힘

```
x_train = train_data['tokenized'].values
y_train = train_data['label'].values
x_test = test_data['tokenized'].values
y_test = test_data['label'].values
```

정수 인코딩

```
t = Tokenizer()
t.fit_on_texts(x_train)
```

```
threshold = 2
total_cnt = len(t.word_index)
rare_cnt = 0
total_freq = 0
rare_freq = 0

for key, value in t.word_counts.items():
    total_freq = total_freq + value

    if (value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

print('단어 집한 (vocabulary)의 크기 :', total_cnt)
print('등장 빈도가 %s번 이하인 희귀단어의 수 : %s'%(threshold-1, rare_cnt))
print('단어 집합에서 희귀단어의 비율 :', (rare_cnt/total_cnt)*100)
print('전체 등장 빈도에서 희귀단어 등장 빈도 비율 :', (rare_freq/total_freq)* 100)
```

```
단어 집한 (vocabulary)의 크기 : 51334
등장 빈도가 1번 이하인 희귀단어의 수 : 27838
단어 집합에서 희귀단어의 비율 : 54.22916585498889
전체 등장 빈도에서 희귀단어 등장 빈도 비율 : 1.162270263951168
```

```
vocab_size = total_cnt - rare_cnt +2
print('단어 집합의 크기 :', vocab_size)
```

```
단어 집합의 크기 : 23498
```

```
original_vocab_size = vocab_size + rare_cnt -2
print('원래 vocab size :', original_vocab_size)
```

```
원래 vocab size : 51334
```

```
tokenizer = Tokenizer(vocab_size, oov_token='OOV')
tokenizer.fit_on_texts(x_train)
x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
```

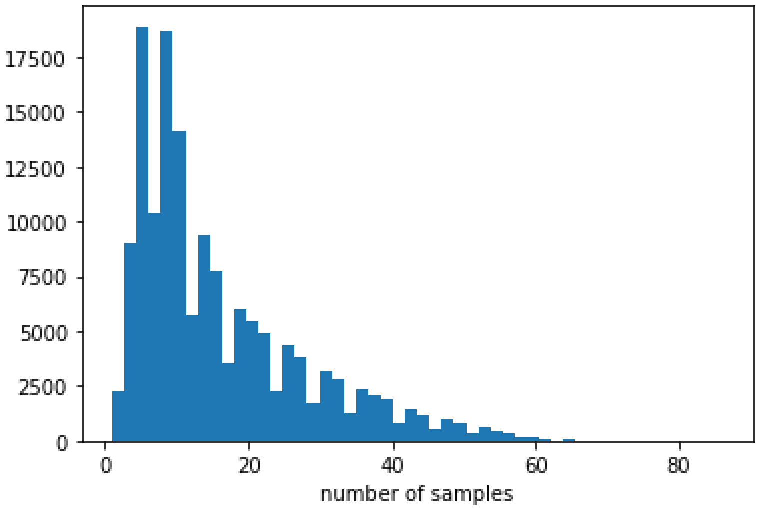
```
print(x_train[:3])
print(x_test[:3])
```

```
[[67, 2086, 302, 14984, 263, 74, 8, 249, 169, 140, 789, 3065, 632, 4, 1], [463
[[16, 696, 2, 755, 116, 2, 193, 252, 14], [340, 3874, 65, 4187, 1639], [13, 71
```

패딩

```
print('리뷰의 최대 길이:', max(len(l) for l in x_train))
print('리뷰의 평균 길이 :', sum(map(len, x_train))/len(x_train))
plt.hist([len(s) for s in x_train], bins=50)
plt.xlabel('length of samples')
plt.xlabel('number of samples')
plt.show()
```

```
리뷰의 최대 길이: 86
리뷰의 평균 길이 : 15.974948476299097
```



```
def below_threshold_len(max_len, nested_list):
    cnt = 0
    for s in nested_list:
        if (len(s) <= max_len):
            cnt = cnt +1
    print('전체 샘플 중 길이가 %s 이하인 샘플의 비율 : %s'%(max_len, (cnt/len(nested_list
```

```
max_len = 80
below_threshold_len(max_len, x_train)
```

```
전체 샘플 중 길이가 80 이하인 샘플의 비율 : 99.99933302652553
```

```
x_train = pad_sequences(x_train, maxlen=max_len)
x_test = pad_sequences(x_test, maxlen=max_len)
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(149931, 80)
(49977, 80)
```

```
from tensorflow.keras.layers import Embedding, Dense, GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

```
# 모델 만들기
```

```
embedding_dim = 100
hidden_size = 128

model_gru = Sequential()
model_gru.add(Embedding(vocab_size, 100))
model_gru.add(GRU(hidden_size))
model_gru.add(Dense(1, activation='sigmoid'))
```

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1,
```

```
model_gru.compile(optimizer='adam', loss = 'binary_crossentropy', metrics=['acc'
history_gru = model_gru.fit(x_train, y_train, epochs=1, callbacks=[es, mc], batc
```

```
2000/2000 [==============================] - ETA: 0s - loss: 0.2863 - acc: 0.8
Epoch 00001: val_acc improved from -inf to 0.90509, saving model to best_model
2000/2000 [==============================] - 50s 24ms/step - loss: 0.2863 - ac
```

```
model_gru.evaluate(x_test, y_test)[1]
```

```
1562/1562 [==============================] - 14s 9ms/step - loss: 0.2518 - acc
0.9045360684394836
```

리뷰 예측하기

```
def sentiment_predict(new_sentence):
    #new_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣]','',new_sentence)
    new_sentence = mecab.morphs(new_sentence)
    new_sentence = [word for word in new_sentence if not word in stopwords]
    encoded = tokenizer.texts_to_sequences([new_sentence])
    pad_new = pad_sequences(encoded, maxlen=max_len)
    score = float(model_gru.predict(pad_new))

    if (score > 0.5): # 긍정
        print("{:.2f}% 확률로 긍정 리뷰 입니다. \n".format(score*100))
    else:
        print("{:.2f}% 확률로 부정 리뷰 입니다. \n".format((1-score)*100))
```

```
sentiment_predict('이 상품은 진짜 너무너무 좋아요!')
```

```
88.03% 확률로 긍정 리뷰 입니다.
```

```
sentiment_predict('이 상품은 진짜 너무너무 별로예요!')
```

```
99.16% 확률로 부정 리뷰 입니다.
```

```
sentiment_predict('이제 수업이 끝난 것 같아요!')
```

```
56.76% 확률로 긍정 리뷰 입니다.
```

```
sentiment_predict('이제 수업이 시작되었어요.')
```

```
78.26% 확률로 긍정 리뷰 입니다.
```