딥러닝 - LSTM 모델을 이용한 가사 장르 예측

이번에는 딥러닝을 사용해 가사 장르를 예측해보겠습니다. 머신러닝때는 내장 벡터화 함수를 사용해봤는데 이번에는 벡터화를 직접 진행해보겠습니다.

먼저 판다스를 임포트해서 노래 데이터를 불러오겠습니다.

In [28]:
```
import pandas as pd
```

In [29]:
```
# songs.csv를 불러와 dataframe으로 만들기
songs = pd.read_csv('songs_preprocessed.csv')
# remove genre == '포크'
songs = songs[songs['genre'] != '포크']
# remove genre == '국악'
songs = songs[songs['genre'] != '국악']
# reset index
songs = songs.reset_index(drop=True)
```

In [30]:
```
songs.head()
```

Out[30]:

	genre	song_id	artist_id	song_name	artist_name	lyric
0	발라드	34431086	839736	취중고백	김민석 (멜로망스)	뭐하고 있었니 늦었지만 잠시 나올래 너의 집 골목에 있는 놀이터에 앉아 있어 친구들...
1	발라드	34481682	261143	겨울잠	아이유	때 이른 봄 몇 송이 꺾어다 너의 방 문 앞에 두었어 긴 잠 실컷 자고 나오면 그때...
2	발라드	34061322	994944	사랑은 늘 도망가	임영웅	눈물이 난다 이 길을 걸으면 그 사람 손길이 자꾸 생각이 난다 붙잡지 못하고 가슴만...
3	발라드	34360855	2138620	눈이 오잖아(Feat.헤이즈)	이무진	한 달 좀 덜 된 기억들 주머니에 넣은 채 걷고 있어 몇 시간을 혹시 몰라 네가 좋...
4	발라드	33496587	2863470	다정히 내 이름을 부르면	경서예지	끝없이 별빛이 내리던 밤 기분 좋은 바람이 두 뺨을 스치고 새벽 바다 한곳을 보는 ...

그 다음은 딥러닝에 사용될 매개변수들을 미리 정의해주겠습니다.

min_words는 가사를 토큰화해서 사전을 만들 때, 최소 몇 번 이상 등장한 단어만을 사용할 것인지를 정의합니다. 이 숫자가 50이라면 최소 50번 이상 등장한 단어만 사용합니다.

lyric_length는 가사를 앞에서 몇 글자만 사용할 것인지를 정의합니다. 이 숫자가 400이라면 400글자 이후 가사는 사용하지 않습니다.

droprate는 데이터를 학습할 때마다 얼마나 많은 데이터를 '빼고 학습'할 지 결정합니다. 높을 수록 과적합을 막는 효과가 있습니다.

In [31]:
```
params ={
  'min_words': 12,
  'lyric_length': 550,
  'droporate': 0.85
}
```

그 다음은 현재 String으로 되어 있는 가사를 정수로 맵핑 해주겠습니다.

In [32]:
```
# 장르 맵핑하기
```

```
songs['genre'] = songs['genre'].map({'발라드':0, '댄스':1, '힙합':2, '트로트':3})
songs.head()
```

Out[32]:

	genre	song_id	artist_id	song_name	artist_name	lyric
0	0	34431086	839736	취중고백	김민석 (멜로망스)	뭐하고 있었니 늦었지만 잠시 나올래 너의 집 골목에 있는 놀이터에 앉아 있어 친구들...
1	0	34481682	261143	겨울잠	아이유	때 이른 봄 몇 송이 꺾어다 너의 방 문 앞에 두었어 긴 잠 실컷 자고 나오면 그때...
2	0	34061322	994944	사랑은 늘 도망가	임영웅	눈물이 난다 이 길을 걸으면 그 사람 손길이 자꾸 생각이 난다 붙잡지 못하고 가슴만...
3	0	34360855	2138620	눈이 오잖아 (Feat.헤이즈)	이무진	한 달 좀 덜 된 기억들 주머니에 넣은 채 걷고 있어 몇 시간을 혹시 몰라 네가 좋...
4	0	33496587	2863470	다정히 내 이름을 부르면	경서예지	끝없이 별빛이 내리던 밤 기분 좋은 바람이 두 뺨을 스치고 새벽 바다 한곳을 보는 ...

In [33]:

```
# na check
songs['lyric']
```

Out[33]:

```
0       뭐하고 있었니 늦었지만 잠시 나올래 너의 집 골목에 있는 놀이터에 앉아 있어 친구들...
1       때 이른 봄 몇 송이 꺾어다 너의 방 문 앞에 두었어 긴 잠 실컷 자고 나오면 그때...
2       눈물이 난다 이 길을 걸으면 그 사람 손길이 자꾸 생각이 난다 붙잡지 못하고 가슴만...
3       한 달 좀 덜 된 기억들 주머니에 넣은 채 걷고 있어 몇 시간을 혹시 몰라 네가 좋...
4       끝없이 별빛이 내리던 밤 기분 좋은 바람이 두 뺨을 스치고 새벽 바다 한곳을 보는 ...
                                ...
1892    내 결 아무리 봐도 난 비단 왜 차이가 날까 너의 시간과 다른 나의 시간 이 밤 지...
1893    하고픈 말이 없어 멍하니 생각이 멈춰 고장 난 듯이 무기력해 두 발이 무거워 가자 ...
1894    baby 옷 입는 것 봐 처음이야 밥 해준 여자 놀라운 유머감각 피식해 야한 농담에...
1895    그 밤에 그 밤 사랑하는 사람들 품으로 그 밤에 그 밤 지나간 추억에 따스함 위로 ...
1896    그대 떠난 여기 노을진 산마루턱엔 아직도 그대 향기가 남아서 이렇게 서있오 나를 두...
Name: lyric, Length: 1897, dtype: object
```

In [34]:

```
# data와 target으로 분류하기
target = songs['genre']
data = songs['lyric']
```

전처리 함수 만들기

In [35]:

```
!pip install konlpy
```

```
Requirement already satisfied: konlpy in /Users/tj/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (0.6.0)
Requirement already satisfied: numpy>=1.6 in /Users/tj/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from konlpy) (1.21.6)
Requirement already satisfied: JPype1>=0.7.0 in /Users/tj/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from konlpy) (1.4.0)
Requirement already satisfied: lxml>=4.1.0 in /Users/tj/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from konlpy) (4.9.1)
Requirement already satisfied: typing-extensions in /Users/tj/opt/anaconda3/envs/tensorflow/lib/python3.7/site-packages (from JPype1>=0.7.0->konlpy) (4.3.0)
```

Okt를 임포트 하는 것 까지는 똑같습니다.

In [36]:

```
from konlpy.tag import Okt
```

```
okt = Okt()
```

In [37]:
```
def preprocess(text):
  txt = text[:params['lyric_length']]
  # 명사만 추출
  return okt.nouns(txt)
  # return okt.morphs(txt)
```

In [38]:
```
data[0]
```

Out[38]:
```
'뭐하고 있었니 늦었지만 잠시 나올래 너의 집 골목에 있는 놀이터에 앉아 있어 친구들 만나서 오랜만에 술을 좀 했는데 자꾸만 니 얼굴 떠올라 무작정 달려왔어 이 맘 모르겠니 요즘 난 미친 사람처럼 너만 생각해 대책없이 네가 점점 좋아져 아냐 안 취했어 진짜야 널 정말 사랑해 눈물이 날만큼 원하고 있어 정말로 몰랐니 가끔 전화해 장난치듯 주말엔 뭐할거냐며 너의 관심 끌던 나를 그리고 한번씩 누나 주려 샀는데 너 그냥 준다고 생색 낸 선물도 너 때문에 산거야 이 맘 모르겠니 요즘 난 미친 사람처럼 너만 생각해 대책없이 네가 점점 좋아져 아냐 안 취했어 진짜야 널 정말 사랑해 진심이야 믿어줘 갑자기 이런 말 놀랐다면 미안해 부담이 되는 게 당연해 이해해 널 하지만 내 고백도 이해해 주겠니 oh 지금 당장 대답하진마 나와 일주일만 사귀어줄래 후회없이 잘 해주고 싶은데 그 후에도 니가 싫다면 나 그때 포기할게 귀찮게 안할게 혼자 아플게 진심이야 너를 사랑하고 있어 '
```

사전 만들기

정의한 Okt 함수(preprocess)를 이용해 data를 토큰화하고, 형태를 확인해보겠습니다.

In [40]:
```
data = [preprocess(lyric) for lyric in data]
data[:1][0][:5]
```

Out[40]:
```
['뭐', '잠시', '너', '집', '골목']
```

토큰의 중복을 방지하면서 동시에 갯수를 세기 위해 dictionary를 만들어줍니다.
그 다음, 뒤에 패딩을 추가하기 위해 가장 긴 토큰 갯수를 확인할 변수 max_length를 만듭니다.

그리고 data의 가사(lyric)들을 순회(loop)하며 max_length를 업데이트 하고
dictionary에 추가하며 갯수를 카운트합니다.
그리고 많이 등장한 순으로 정렬하겠습니다.

In [41]:
```
# 중복 없는 토큰 갯수 파악
tokens = {}
# 가장 긴 단어 갯수 확인(padding 추가 위함)
max_length = 0

for lyric in data:
  if len(lyric) > max_length:
    max_length = len(lyric)
  for token in lyric:
    if token not in tokens:
      tokens[token] = 0
    tokens[token] += 1
tokens = sorted(tokens.items(), key = lambda x:x[1], reverse = True)
```

In [42]:
```
len(tokens)
```

Out[42]:
```
7381
```

정수로 변환

이번에도 두 작업을 동시에 진행하겠습니다.
우리는 어떤 토큰이 어떤 정수로 변환되었는지 기록해두어야 하기 때문에(재현성) index로 맵핑하기 위한 dictionary를 추가하고
일정 횟수 이상 등장한 단어만을 단어장에 추가하기 위해 params['min_words']보다 등장 빈도가 많은지 체크합니다.
이 값은 맨 처음 우리가 임의로 설정한 파라미터입니다.

In [43]:
```
token_to_index = {}
i = 1
max_i = 0
for (token, frequency) in tokens:
  if frequency > params['min_words']:
    token_to_index[token] = i
    i += 1
    max_i = i
  else:
    token_to_index[token] = 0 # 빈도수 1이면 0으로 바꿔버린다
```

그 다음 json파일로 dictionary를 저장합니다. 이 dictionary는 예측할 때 사용합니다.
유저가 특정 가사를 입력하면, 이 dictionary대로 정수로 변환하고, 정수로 변환된 array를 딥러닝 모델에 집어넣어 장르를 예측합니다.

In [44]:
```
token_to_index
# to save as json
import json
# utf-8, no index
with open('token_to_index.json', 'w', encoding='utf-8') as f:
    json.dump(token_to_index, f, ensure_ascii=False)
```

그 다음 data에 있는 토큰화된 lyric(가사)들을 정수로 맵핑해주겠습니다.

In [45]:
```
data_indexed = [[token_to_index[token] for token in lyric] for lyric in data]
```

패딩 추가하기

지금은 lyric마다 토큰화, 정수화 된 길이가 다릅니다.
길이를 통일하기 위해 가장 긴 길이를 가진 가사를 기준으로 패딩을 추가합니다.

In [46]:
```
from tensorflow import keras
import tensorflow as tf
import numpy as np
```

In [47]:
```
# random seed for deep learning
seed = 42
tf.random.set_seed(seed)
np.random.seed(seed)
```

In [48]:
```
# max_length 미달인 녀석들 0으로 패딩
data_padded = keras.preprocessing.sequence.pad_sequences(data_indexed, maxlen
```

데이터를 확인해보면 post 방법으로 padding을 추가했기 때문에, 데이터의 뒷쪽에 0들이 추가된 것을 볼 수 있습니다.

In [49]:
```
data_padded
```

Out[49]:
```
array([[ 59, 174,   2, ...,   0,   0,   0],
       [ 20, 185, 180, ...,   0,   0,   0],
       [ 39,  10,  67, ...,   0,   0,   0],
       ...,
       [358,  15,  97, ...,   0,   0,   0],
       [  6,  21,   6, ...,   0,   0,   0],
       [  9,  81, 446, ...,   0,   0,   0]], dtype=int32)
```

In [50]:
```
data_padded[0].shape
```

Out[50]:
```
(141,)
```

train test split

In [51]:
```
import numpy as np
from sklearn.model_selection import train_test_split
```

In [52]:
```
data_final = np.array(data_padded)
target_final = np.array(target)

train_X , test_X , train_y , test_y = train_test_split(data_final , target_fi
train_X , valid_X , train_y , valid_y = train_test_split(train_X , train_y ,
```

모델 구현

모델을 구현해보겠씁니다. 대부분 하이퍼파라미터들인데, mask_zero = True에 체크해 주는 것 정도가 가장 중요합니다.
이 설정은 패딩값으로 입력된 0이라는 숫자를 딥러닝 시 무시하게 합니다.
LSTM 층 다음 0.85의 드롭아웃 층, relu 층 다음 또 0.85의 드롭아웃 층, 마지막으로 softmax 덴스 층을 추가했습니다.
몇 개의 층을 사용하고 어떤 층을 추가할 지는 분석가가 결정합니다.

In [53]:
```
model = keras.Sequential()
model.add(keras.layers.Embedding(
    input_dim = len(tokens) + 1,
    output_dim = 32,
    input_length = max_length,
    mask_zero = True
))
model.add(keras.layers.LSTM(128))
model.add(keras.layers.Dropout(params['droporate']))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(params['droporate']))
model.add(keras.layers.Dense(4, activation='softmax'))
model.summary()
```

```
2022-08-09 16:00:07.630065: I tensorflow/core/platform/cpu_feature_guard.cc:19
3] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
```

```
(oneDNN) to use the following CPU instructions in performance-critical operati
ons:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate co
mpiler flags.
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 141, 32)           236224

 lstm (LSTM)                 (None, 128)               82432

 dropout (Dropout)           (None, 128)               0

 dense (Dense)               (None, 64)                8256

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 4)                 260

=================================================================
Total params: 327,172
Trainable params: 327,172
Non-trainable params: 0
_________________________________________________________________
```

만들어진 모델을 컴파일하고 훈련합니다.

모델의 성능은 에포크가 진행됨에 따라 좋아졌다가 나빠지기도 합니다. 가장 좋은 상태를 저장할 수 있도록 checkpoint를 추가합니다.

In [54]:

```
model.compile(
  optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'],
)
checkpoint_cb = keras.callbacks.ModelCheckpoint(
  'song_genre_predict.h5',
  save_best_only=True
)
hist = model.fit(
  train_X,
  train_y,
  epochs=30,
  validation_data=(valid_X, valid_y),
  callbacks=[checkpoint_cb],
)
```

```
Epoch 1/30
38/38 [==============================] - 11s 157ms/step - loss: 1.3889 - accur
acy: 0.2440 - val_loss: 1.3784 - val_accuracy: 0.3651
Epoch 2/30
38/38 [==============================] - 4s 117ms/step - loss: 1.3780 - accura
cy: 0.2861 - val_loss: 1.3727 - val_accuracy: 0.3651
Epoch 3/30
38/38 [==============================] - 4s 110ms/step - loss: 1.3729 - accura
cy: 0.2836 - val_loss: 1.3601 - val_accuracy: 0.3651
Epoch 4/30
38/38 [==============================] - 4s 112ms/step - loss: 1.3408 - accura
cy: 0.3471 - val_loss: 1.2605 - val_accuracy: 0.4276
Epoch 5/30
38/38 [==============================] - 4s 112ms/step - loss: 1.3118 - accura
cy: 0.3710 - val_loss: 1.2354 - val_accuracy: 0.4079
Epoch 6/30
```

```
38/38 [==============================] - 4s 114ms/step - loss: 1.2178 - accura
cy: 0.4270 - val_loss: 1.1632 - val_accuracy: 0.4112
Epoch 7/30
38/38 [==============================] - 4s 109ms/step - loss: 1.1475 - accura
cy: 0.4353 - val_loss: 1.1660 - val_accuracy: 0.4474
Epoch 8/30
38/38 [==============================] - 4s 111ms/step - loss: 1.0851 - accura
cy: 0.4782 - val_loss: 1.1678 - val_accuracy: 0.4803
Epoch 9/30
38/38 [==============================] - 4s 111ms/step - loss: 1.0709 - accura
cy: 0.4823 - val_loss: 1.1472 - val_accuracy: 0.4737
Epoch 10/30
38/38 [==============================] - 4s 112ms/step - loss: 1.0234 - accura
cy: 0.5103 - val_loss: 1.0969 - val_accuracy: 0.5757
Epoch 11/30
38/38 [==============================] - 4s 111ms/step - loss: 0.9714 - accura
cy: 0.5433 - val_loss: 0.9785 - val_accuracy: 0.6250
Epoch 12/30
38/38 [==============================] - 4s 114ms/step - loss: 0.8954 - accura
cy: 0.5870 - val_loss: 0.9676 - val_accuracy: 0.6447
Epoch 13/30
38/38 [==============================] - 4s 112ms/step - loss: 0.8837 - accura
cy: 0.5985 - val_loss: 1.1616 - val_accuracy: 0.4803
Epoch 14/30
38/38 [==============================] - 4s 109ms/step - loss: 0.9212 - accura
cy: 0.5862 - val_loss: 1.0630 - val_accuracy: 0.5691
Epoch 15/30
38/38 [==============================] - 4s 109ms/step - loss: 0.8063 - accura
cy: 0.6496 - val_loss: 0.9769 - val_accuracy: 0.5855
Epoch 16/30
38/38 [==============================] - 4s 110ms/step - loss: 0.7682 - accura
cy: 0.6406 - val_loss: 1.0314 - val_accuracy: 0.6513
Epoch 17/30
38/38 [==============================] - 4s 112ms/step - loss: 0.7071 - accura
cy: 0.6620 - val_loss: 0.9793 - val_accuracy: 0.6118
Epoch 18/30
38/38 [==============================] - 4s 110ms/step - loss: 0.6474 - accura
cy: 0.6917 - val_loss: 1.1450 - val_accuracy: 0.6612
Epoch 19/30
38/38 [==============================] - 4s 109ms/step - loss: 0.6413 - accura
cy: 0.6917 - val_loss: 1.0260 - val_accuracy: 0.6776
Epoch 20/30
38/38 [==============================] - 4s 109ms/step - loss: 0.6226 - accura
cy: 0.6884 - val_loss: 1.1493 - val_accuracy: 0.6250
Epoch 21/30
38/38 [==============================] - 4s 109ms/step - loss: 0.5699 - accura
cy: 0.7057 - val_loss: 1.3718 - val_accuracy: 0.5954
Epoch 22/30
38/38 [==============================] - 4s 110ms/step - loss: 0.5699 - accura
cy: 0.7123 - val_loss: 1.3435 - val_accuracy: 0.6349
Epoch 23/30
38/38 [==============================] - 4s 109ms/step - loss: 0.5093 - accura
cy: 0.7205 - val_loss: 2.0193 - val_accuracy: 0.6053
Epoch 24/30
38/38 [==============================] - 4s 112ms/step - loss: 0.6164 - accura
cy: 0.7090 - val_loss: 1.2617 - val_accuracy: 0.6447
Epoch 25/30
38/38 [==============================] - 4s 109ms/step - loss: 0.7001 - accura
cy: 0.6529 - val_loss: 1.1141 - val_accuracy: 0.5099
Epoch 26/30
38/38 [==============================] - 4s 111ms/step - loss: 0.6433 - accura
cy: 0.6810 - val_loss: 1.3807 - val_accuracy: 0.4901
Epoch 27/30
38/38 [==============================] - 4s 110ms/step - loss: 0.7396 - accura
```

```
cy: 0.6620 - val_loss: 1.2266 - val_accuracy: 0.5888
Epoch 28/30
38/38 [==============================] - 4s 111ms/step - loss: 0.5953 - accura
cy: 0.7115 - val_loss: 1.3976 - val_accuracy: 0.6217
Epoch 29/30
38/38 [==============================] - 4s 111ms/step - loss: 0.5568 - accura
cy: 0.7453 - val_loss: 1.2823 - val_accuracy: 0.6217
Epoch 30/30
38/38 [==============================] - 4s 112ms/step - loss: 0.4975 - accura
cy: 0.7255 - val_loss: 1.7232 - val_accuracy: 0.6086
```

checkpoint를 통해 저장한 가장 좋은 모델을 불러옵니다.

In [55]:

```
model = keras.models.load_model('song_genre_predict.h5')
```

시각화

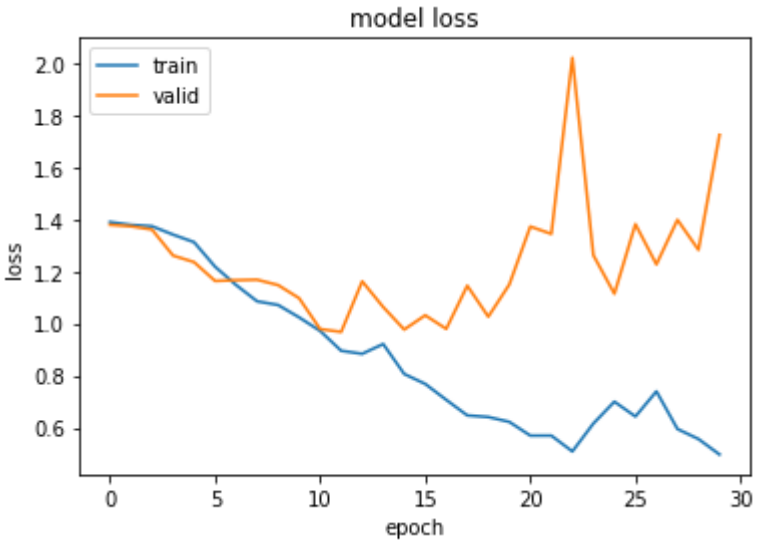
학습결과를 시각화해 확인해보겠습니다.

epoch 진행에 따른 validation loss와 loss를 비교해보고, validation accuracy와 accuracy를 비교해보겠습니다.

In [56]:

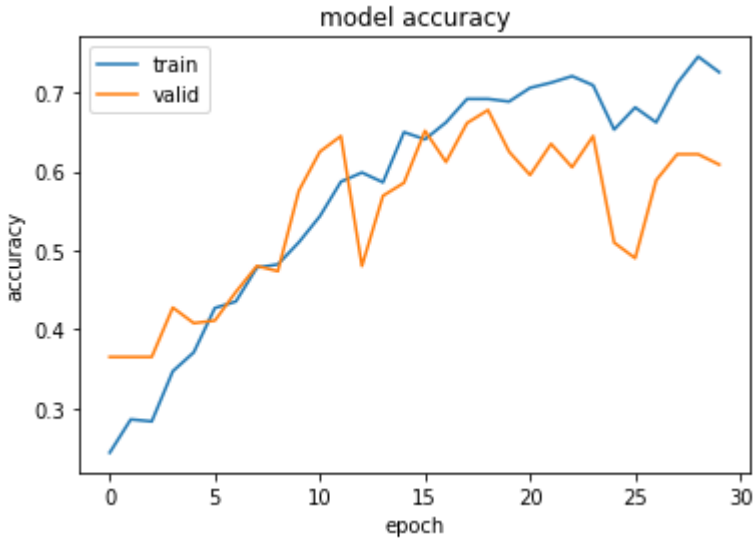
```
import matplotlib.pyplot as plt

# draw graph
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



In [57]:

```
plt.plot(hist.history['accuracy'])
plt.plot(hist.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'valid'], loc='upper left')
plt.show()
```



전체 데이터로 모델 다시 학습시키기

이제 어떤 레이어를 사용하고 어떤 파라미터를 사용할지 결정되었으니, 모델을 다시 학습시키겠습니다.

타겟의 이름은 target이고, 토큰화와 패딩이 완료된 전체 데이터는 data_padded 변수에 저장되어 있습니다.

In [58]:
```
# print(target[:3])
# print(data_padded[:3])
# print(data_padded[0].shape)
```

train-test split 대신, train-validation split을 진행하겠습니다.

In [59]:
```
train_X, valid_X, train_y, valid_y = train_test_split(data_padded, target, te
```

모델도 아래와 같이 새로 만들어줍니다.

In [60]:
```
model = keras.Sequential()
model.add(keras.layers.Embedding(
    input_dim = len(tokens) + 1,
    output_dim = 32,
    input_length = max_length,
    mask_zero = True
))
model.add(keras.layers.LSTM(128))
model.add(keras.layers.Dropout(params['droporate']))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dropout(params['droporate']))
model.add(keras.layers.Dense(4, activation='softmax'))
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 141, 32)           236224

 lstm_1 (LSTM)                (None, 128)               82432

 dropout_2 (Dropout)          (None, 128)               0

 dense_2 (Dense)              (None, 64)                8256

 dropout_3 (Dropout)          (None, 64)                0
```

```
 dense_3 (Dense)              (None, 4)                 260

=================================================================
Total params: 327,172
Trainable params: 327,172
Non-trainable params: 0
_________________________________________________________________
```

모델을 컴파일하고 다시 훈련하겠습니다.

In [61]:

```
model.compile(
  optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'],
)
checkpoint_cb = keras.callbacks.ModelCheckpoint(
  'song_genre_predict.h5',
  save_best_only=True
)
hist = model.fit(
  train_X,
  train_y,
  epochs=20,
  validation_data=(valid_X, valid_y),
  callbacks=[checkpoint_cb]
)
```

```
Epoch 1/20
48/48 [==============================] - 11s 138ms/step - loss: 1.3821 - accur
acy: 0.2999 - val_loss: 1.3833 - val_accuracy: 0.2684
Epoch 2/20
48/48 [==============================] - 5s 105ms/step - loss: 1.3821 - accura
cy: 0.2914 - val_loss: 1.3802 - val_accuracy: 0.2684
Epoch 3/20
48/48 [==============================] - 5s 111ms/step - loss: 1.3724 - accura
cy: 0.2966 - val_loss: 1.3677 - val_accuracy: 0.2684
Epoch 4/20
48/48 [==============================] - 5s 107ms/step - loss: 1.3610 - accura
cy: 0.3131 - val_loss: 1.3403 - val_accuracy: 0.3316
Epoch 5/20
48/48 [==============================] - 5s 112ms/step - loss: 1.2862 - accura
cy: 0.3869 - val_loss: 1.1886 - val_accuracy: 0.4342
Epoch 6/20
48/48 [==============================] - 5s 105ms/step - loss: 1.1945 - accura
cy: 0.4278 - val_loss: 1.2465 - val_accuracy: 0.3974
Epoch 7/20
48/48 [==============================] - 6s 115ms/step - loss: 1.1666 - accura
cy: 0.4529 - val_loss: 1.1372 - val_accuracy: 0.4395
Epoch 8/20
48/48 [==============================] - 5s 107ms/step - loss: 1.0850 - accura
cy: 0.4964 - val_loss: 1.1298 - val_accuracy: 0.4316
Epoch 9/20
48/48 [==============================] - 5s 107ms/step - loss: 1.0873 - accura
cy: 0.4891 - val_loss: 1.1859 - val_accuracy: 0.4053
Epoch 10/20
48/48 [==============================] - 5s 107ms/step - loss: 1.0404 - accura
cy: 0.5274 - val_loss: 0.9855 - val_accuracy: 0.5974
Epoch 11/20
48/48 [==============================] - 5s 108ms/step - loss: 1.0146 - accura
cy: 0.5478 - val_loss: 1.0505 - val_accuracy: 0.5526
Epoch 12/20
48/48 [==============================] - 5s 105ms/step - loss: 0.9220 - accura
cy: 0.5887 - val_loss: 1.0774 - val_accuracy: 0.6053
```

```
Epoch 13/20
48/48 [==============================] - 5s 105ms/step - loss: 0.9599 - accura
cy: 0.5926 - val_loss: 1.0114 - val_accuracy: 0.5947
Epoch 14/20
48/48 [==============================] - 5s 106ms/step - loss: 0.8293 - accura
cy: 0.6447 - val_loss: 1.0240 - val_accuracy: 0.5526
Epoch 15/20
48/48 [==============================] - 5s 107ms/step - loss: 0.8148 - accura
cy: 0.6341 - val_loss: 1.0182 - val_accuracy: 0.5868
Epoch 16/20
48/48 [==============================] - 5s 106ms/step - loss: 0.7214 - accura
cy: 0.6717 - val_loss: 1.1716 - val_accuracy: 0.5842
Epoch 17/20
48/48 [==============================] - 5s 105ms/step - loss: 0.7322 - accura
cy: 0.6599 - val_loss: 1.0420 - val_accuracy: 0.5711
Epoch 18/20
48/48 [==============================] - 5s 106ms/step - loss: 0.6560 - accura
cy: 0.6882 - val_loss: 1.2224 - val_accuracy: 0.5868
Epoch 19/20
48/48 [==============================] - 5s 105ms/step - loss: 0.6790 - accura
cy: 0.6836 - val_loss: 1.2598 - val_accuracy: 0.6000
Epoch 20/20
48/48 [==============================] - 5s 107ms/step - loss: 0.6510 - accura
cy: 0.6987 - val_loss: 1.1727 - val_accuracy: 0.5684
```

예측해보기

실제로 모델을 이용해 예측해보겠습니다.

input_lyric에 학습한 적 없는 임의의 가사를 적고, 그 내용을 예측합니다.

In [62]:

```
# 예측해보기
inputLyric = '오케이 난 술 투더 제이 프리스타일 랩 배틀 챔피언 불쌍해 또 투덜'
# read token_to_index from csv with dict
token_to_index_df = pd.read_csv('token_to_index.csv', index_col=0)

# tokenize inputlyric
input_indexed = okt.nouns(inputLyric)
print(input_indexed)

input_numbered = []

# token to index
for token in input_indexed:
  if token in token_to_index_df.index:
    input_numbered.append(token_to_index_df.loc[token][0])

# padding to 105
input_indexed_indexed_padded = keras.preprocessing.sequence.pad_sequences([in
print(input_indexed_indexed_padded)
# pred
pred = model.predict(np.array([input_indexed_indexed_padded]))
print(pred)

genre_list = ['발라드', '댄스', '힙합', '트로트']

# argmax to get genre
pred_index = np.argmax(pred)
pred_genre = genre_list[pred_index]
print(pred_genre)
```

```
['오케이', '난', '술', '투더', '제이', '프리스타일', '랩', '배틀', '챔피언', '또', '투덜']
[  0   5 136   0  23   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   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]
1/1 [==============================] - 2s 2s/step
[[0.1953446  0.39751786 0.35106537 0.05607211]]
댄스
```