개인 프로젝트

스팸 메일 분류

발표자 : 김성윤

목차

1. 프로젝트 목표

- 딥러닝을 활용한 이진 분류 문제 해결하기
- 성능에 영향을 미치는 요소들을 찾고 성능 높이기

2. 모듈·데이터 불러오기 & 데이터 확인

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from keras_preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, BatchNormalization, Embedding, LSTM, GRU
from keras import callbacks
import matplotlib.pyplot as plt
from sklearn.metrics import f1_score
import tensorflow as tf
import keras.backend as K
```

```
#f1 score 학습 지표 함수

def f1(y_true, y_pred):
  y_pred = K.round(y_pred)
  tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
  tn = K.sum(K.cast((1-y_true)*(1-y_pred), 'float'), axis=0)
  fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
  fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

  p = tp / (tp + fp + K.epsilon())
  r = tp / (tp + fn + K.epsilon())

  f1 = 2*p*r / (p+r+K.epsilon())
  "f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)"
  return K.mean(f1)
```

```
[5]  dt = pd.read_csv('./SPAM_text_message.csv')
```

```
[6]  dt.sample(5)
```

3. 데이터 전처리

[5]
0초

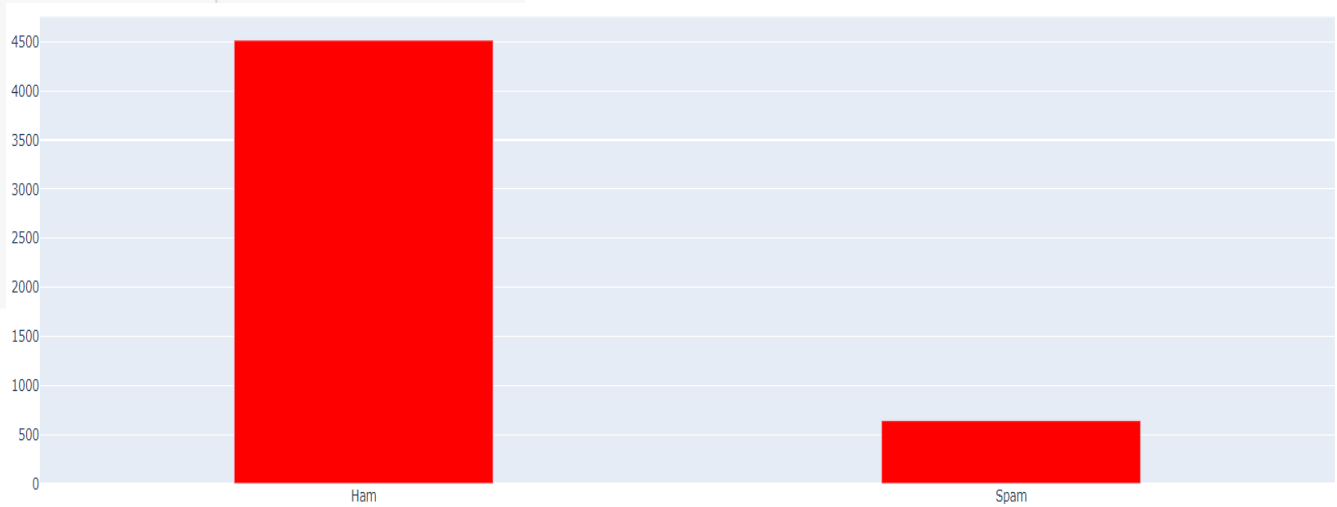
```
# 결측치 확인
print('결측값 여부 :',dt.isnull().values.any())

# 데이터 중복 확인
print('Message 중 중복이 아닌 값 :',dt['Message'].nunique())

# 중복 데이터 제거
dt.drop_duplicates(subset=['Message'], inplace=True)

# 제거되었는지 점검
print("중복 제거 후 데이터 크기 :", len(dt))
```

```
결측값 여부 : False
Message 중 중복이 아닌 값 : 5157
중복 제거 후 데이터 크기 : 5157
```

3. 데이터 전처리

```
[6]  #레이블 비율 확인
     dt.replace('ham', 0, inplace=True)
     dt.replace('spam', 1, inplace=True)

     print(f'전체 데이터 중 정상 메일의 비율 = {round(dt["Category"].value_counts()[0]/len(dt) * 100,2)}%')
     print(f'전체 데이터 중 스팸 메일의 비율 = {round(dt["Category"].value_counts()[1]/len(dt) * 100,2)}%')

     groups = dt.groupby(by='Category').count().Message
     CNT_HAM = groups[0]
     CNT_SPAM = groups[1]
     fig = go.Figure()
     fig.add_trace(go.Bar(
         x=['Ham', 'Spam'],
         y=[CNT_HAM, CNT_SPAM],
         marker_color='red',
         width=[0.4,0.4]))
```

```
전체 데이터 중 정상 메일의 비율 = 87.57%
전체 데이터 중 스팸 메일의 비율 = 12.43%
```



=> 성능 지표로 f1_score 사용 (f1_score는 정밀도, 재현율 중 하나가 0에 가깝거나 두 값이 비슷할 때에 높아짐)

3. 데이터 전처리

```
[7]  X_data = dt['Message']
     y_data = dt['Category']

     # 정수 인코딩
     NUM_WORDS = 10000

     tokenizer = Tokenizer(num_words=NUM_WORDS)
     tokenizer.fit_on_texts(X_data)

     word_to_index = tokenizer.word_index
     print(word_to_index)
```

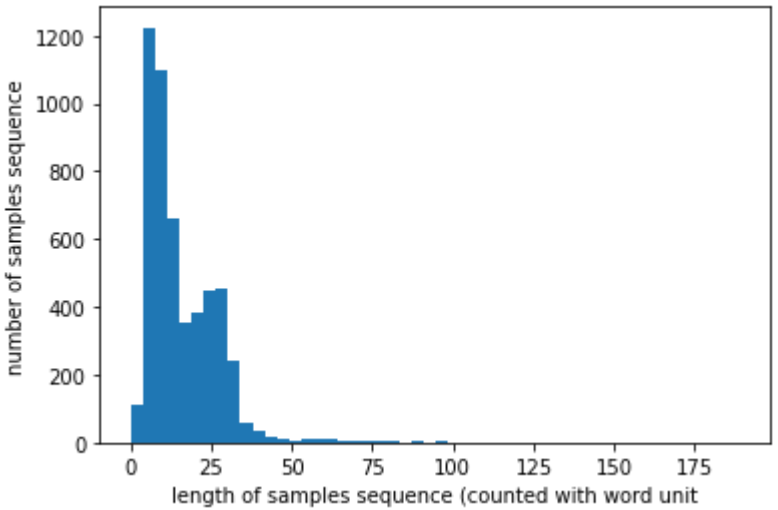
0초

```
{'i': 1, 'to': 2, 'you': 3, 'a': 4, 'the': 5, 'u': 6, 'and': 7, 'in': 8, 'is': 9, 'me': 10, 'my': 11, 'for': 12,
```

3. 데이터 전처리

```
#워드 임베딩
X_sequences_data = tokenizer.texts_to_sequences(X_data)
```

```
# 단어 단위의 길이 계산
plt.hist([len(sample) for sample in X_sequences_data], bins=50)
plt.xlabel('length of samples sequence (counted with word unit')
plt.ylabel('number of samples sequence')
plt.show()
```




```
메일의 최대 길이 : 189
메일의 평균 길이 : 15.680628
```

```
#최대 길이 기준으로 패딩
MAX_TEXT_LEN = 189
X = pad_sequences(X_sequences_data, maxlen=MAX_TEXT_LEN)
y = y_data.copy()

# 데이터 분리
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True, random_state=42, stratify = y)
```

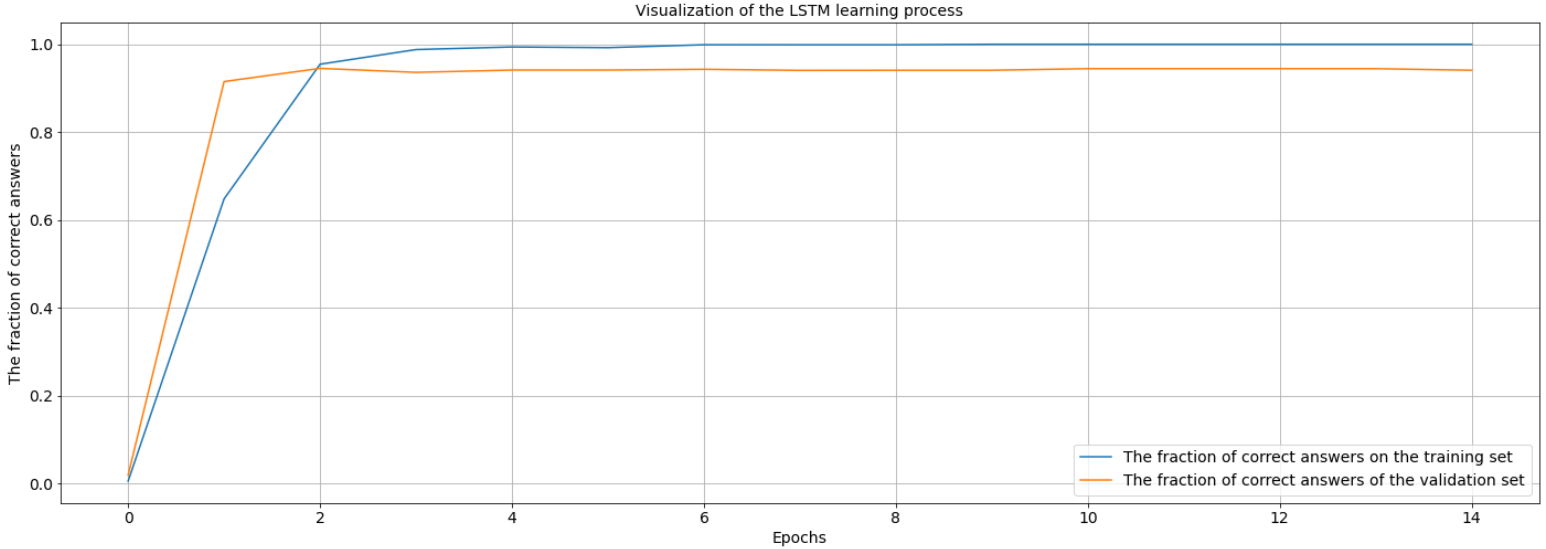
4. 모델링 – LSTM 사용

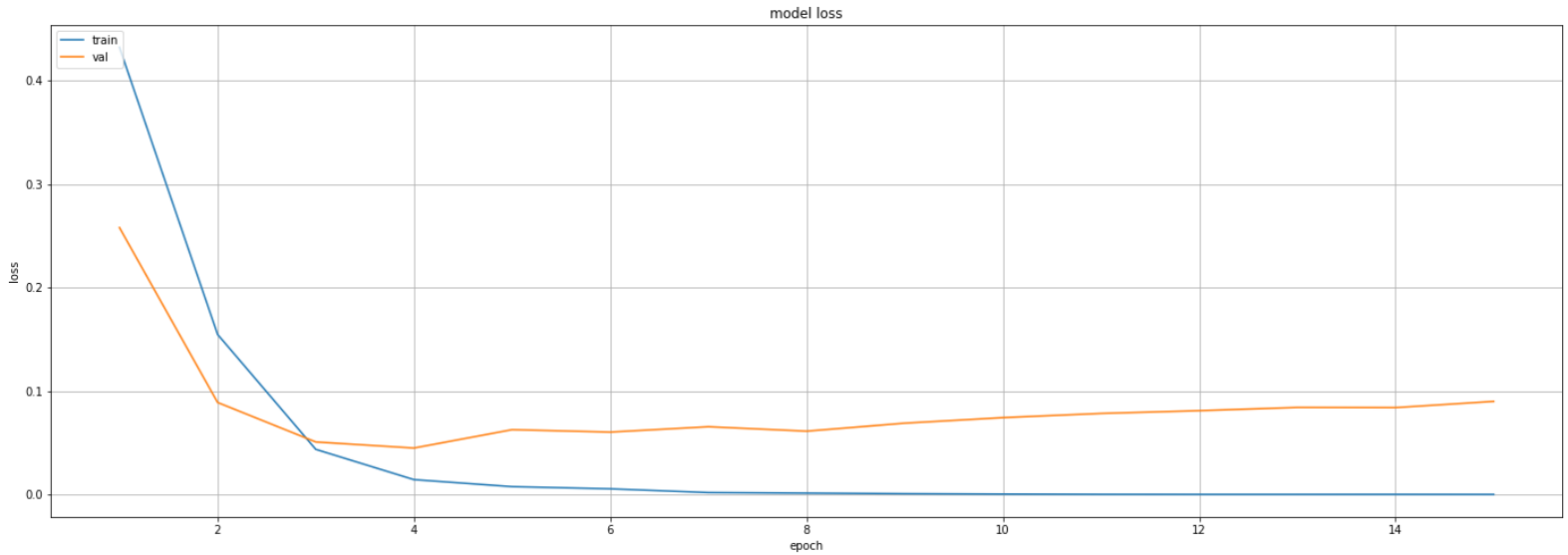
```
# 모델링
vocab_size = 10000

model = Sequential()
model.add(Embedding(vocab_size, 128, input_length=MAX_TEXT_LEN))
model.add(LSTM(64, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(64,activation='relu'))
model.add(Dense(1, activation='sigmoid'))

callbacks_list = [
    callbacks.EarlyStopping(monitor='f1', min_delta=0.01, patience=15, verbose=1, mode = 'auto'),
    callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, min_delta=0.01, min_lr=1e-10, patience=15, verbose=1, mode='auto')
]
# ReduceLROnPlateau : 모델의 개선이 없을 경우, learning rate를 조절하는 장치
```

```
[17] """model.compile(metrics=['Accuracy'], loss='binary_crossentropy', optimizer='Adam')"""
     model.compile(metrics=[ f1 ], loss='binary_crossentropy', optimizer='Adam')
```

```
history = model.fit(X_train, y_train, batch_size=128, epochs=15, validation_split=0.2,  callbacks=callbacks_list)
```

4. 모델링 – LSTM 사용


Visualization of the LSTM learning process
The fraction of correct answers
Epochs
The fraction of correct answers on the training set
The fraction of correct answers of the validation set


model loss
train
val
loss
epoch

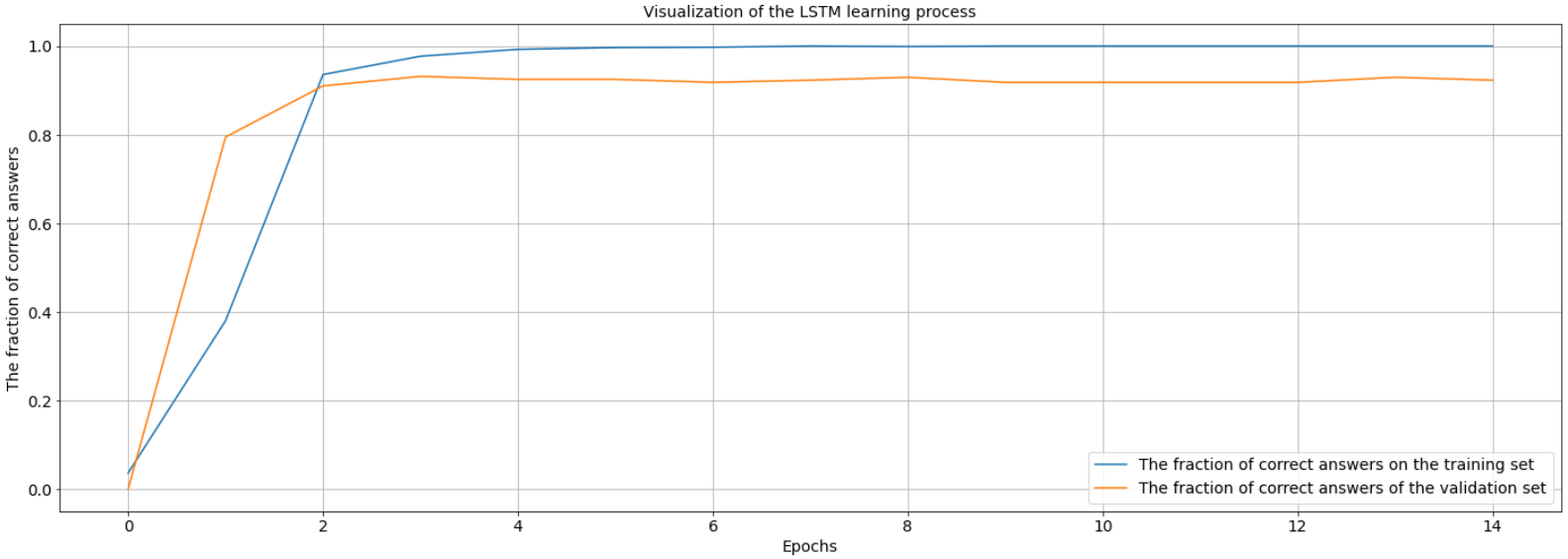
4. 모델링 – GRU 사용

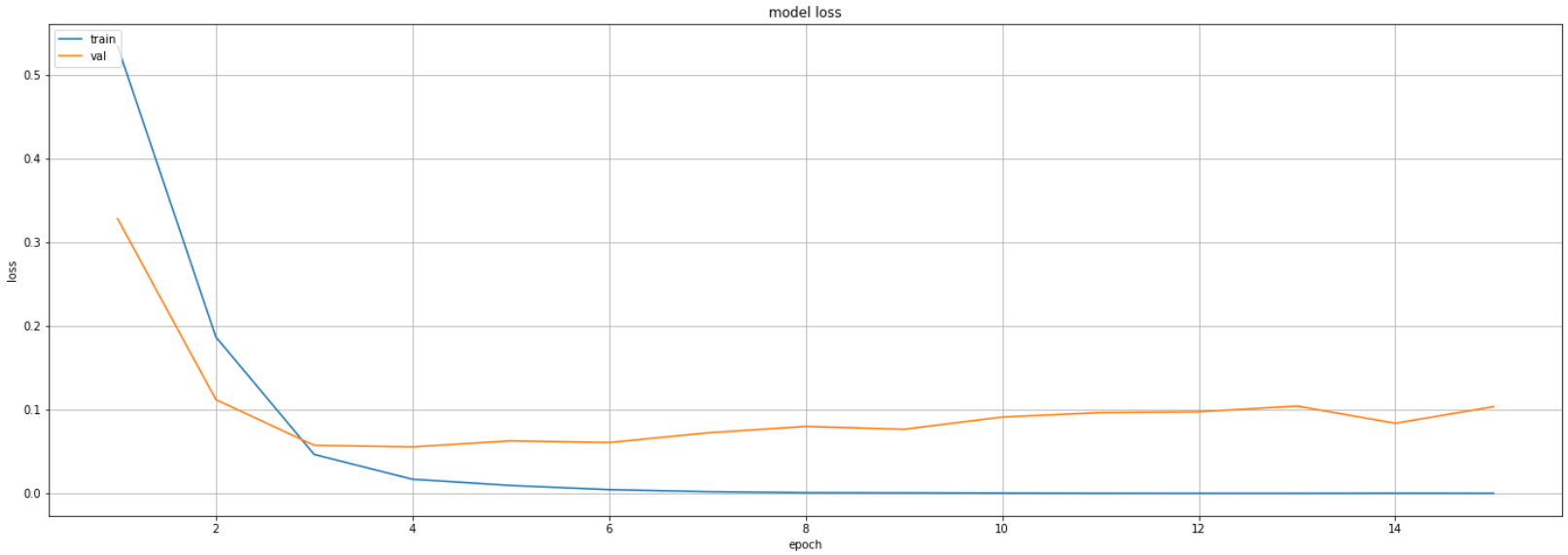
```
# 모델링
vocab_size = 10000

model = Sequential()
model.add(Embedding(vocab_size, 128, input_length=MAX_TEXT_LEN))
model.add(GRU(64, dropout=0.3, recurrent_dropout=0.3))
model.add(Dense(64,activation='relu'))
model.add(Dense(1, activation='sigmoid'))

callbacks_list = [
    callbacks.EarlyStopping(monitor='f1', min_delta=0.01, patience=15, verbose=1, mode = 'auto'),
    callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, min_delta=0.01, min_lr=1e-10, patience=15, verbose=1, mode='auto')
]
# ReduceLROnPlateau : 모델의 개선이 없을 경우, learning rate를 조절하는 장치
```

```
[17] """model.compile(metrics=['Accuracy'], loss='binary_crossentropy', optimizer='Adam')"""
     model.compile(metrics=[ f1 ], loss='binary_crossentropy', optimizer='Adam')
```

```
history = model.fit(X_train, y_train, batch_size=128, epochs=15, validation_split=0.2,  callbacks=callbacks_list)
```

4. 모델링 – GRU 사용


Visualization of the LSTM learning process
The fraction of correct answers
Epochs
The fraction of correct answers on the training set
The fraction of correct answers of the validation set


model loss
train
val
loss
epoch

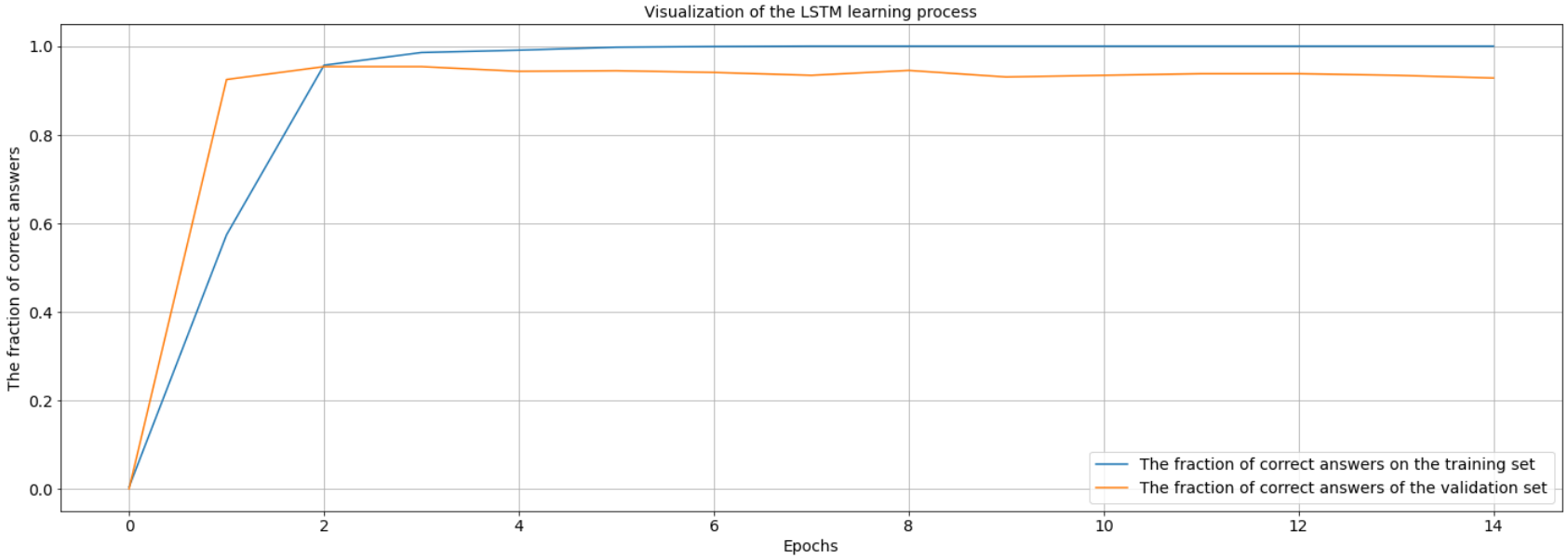
4. 모델링 – BiLSTM 사용

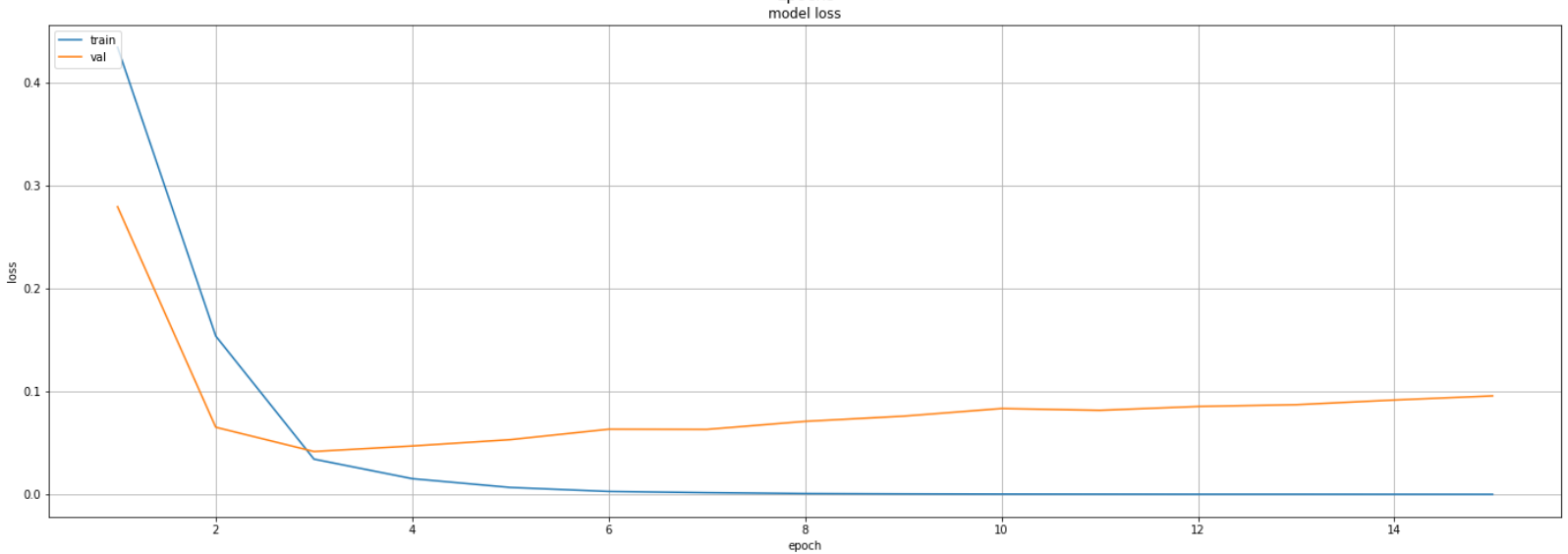
```
# 모델링
vocab_size = 10000

model = Sequential()
model.add(Embedding(vocab_size, 128, input_length=MAX_TEXT_LEN))
model.add(Bidirectional(LSTM(64, dropout=0.3, recurrent_dropout=0.3)))
model.add(Dense(64,activation='relu'))
model.add(Dense(1, activation='sigmoid'))

callbacks_list = [
    callbacks.EarlyStopping(monitor='f1', min_delta=0.01, patience=15, verbose=1, mode = 'auto'),
    callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, min_delta=0.01, min_lr=1e-10, patience=15, verbose=1, mode='auto')
]
# ReduceLROnPlateau : 모델의 개선이 없을 경우, learning rate를 조절하는 장치
```

```
[17] """model.compile(metrics=['Accuracy'], loss='binary_crossentropy', optimizer='Adam')"""
     model.compile(metrics=[ f1 ], loss='binary_crossentropy', optimizer='Adam')
```

```
history = model.fit(X_train, y_train, batch_size=128, epochs=15, validation_split=0.2,  callbacks=callbacks_list)
```

4. 모델링 – BiLSTM 사용





5. 결과 확인

평가 / 모델	F1 score	Model.evaluate
LSTM	0.9262	0.9207
GRU	0.9153	0.9057
BiLSTM	0.9300	0.9250

6. 결론

- Keras에서 제공하는 대표적인 모델인 LSTM, GRU, BiLSTM을 사용
- BiLSTM > LSTM > GRU 순으로 성능이 높았지만, 세 모델 모두 LSTM 기반인 점에서 확연한 차이를 보이지는 않고 비슷했음
- 모델보다는 하이퍼파라미터와 layer가 성능에 큰 영향을 미침

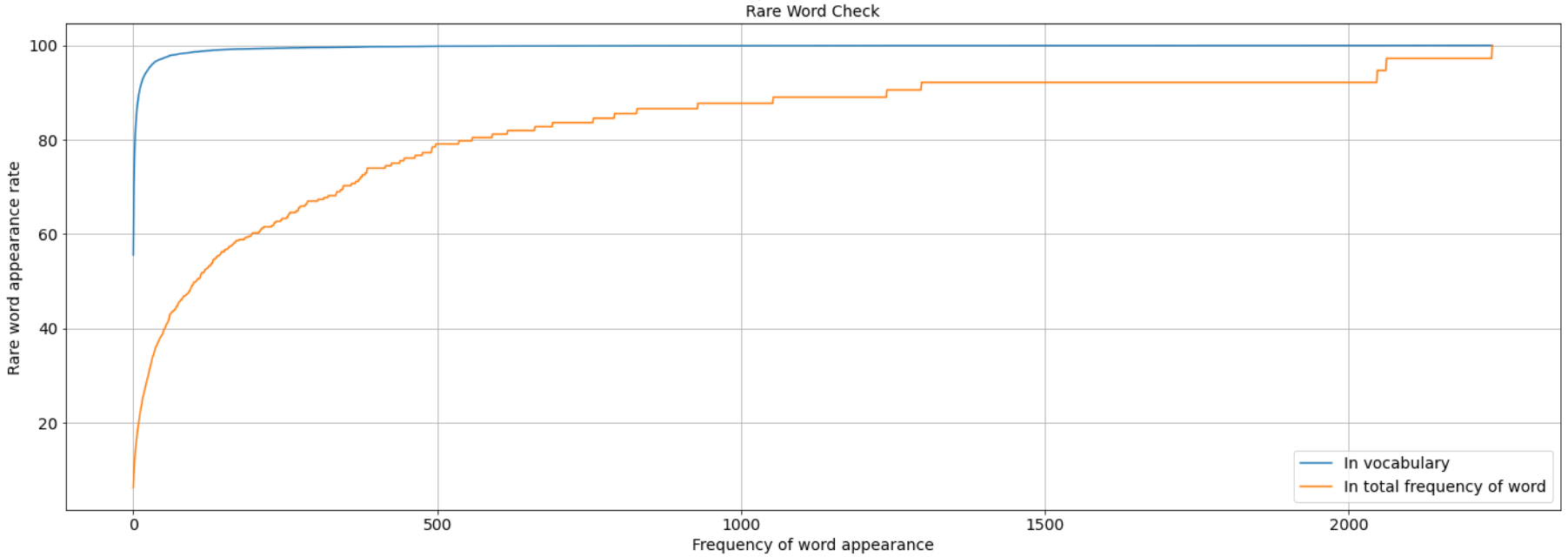
- 모델 학습 과정에서 Batchsize의 크기가 작을 경우, 특정 epoch에서 성능이 달라지는 문제가 발생
- Batchsize=64 -> batchsize=128로 키웠을 때, 더 이상 위의 문제가 발생하지 않음 (값이 보정됨)

- 모델을 스태킹하여 둘 이상의 layer를 쌓기보다 하나의 layer가 더 성능이 높았음 (복잡도만 높임)
- unit은 모델이 데이터를 더 잘 설명할 수 있게 만드는 것인데, 이 데이터에서는 큰 영향을 미치지 않고 학습 속도에만 눈에 띄게 영향을 미침

- Dropout 혹은 recurrent_dropout이 없을 경우 과적합으로 인해서 val_loss값도 계속 낮아짐
- Dropout 값은 보편적으로 사용하는 0.3 사용

7. 한계점

- 배치 정규화 층을 추가했을 때, 성능이 오히려 떨어지는 문제가 발생
- Transformer와 같은 실용적인 모델을 적용해보지 못함
- 데이터 정제 과정에서 threshold(등장 빈도 수) 값을 정하는 기준을 찾지 못함
- 문제와 데이터의 특징에 따라 전처리 과정에서 불용어를 제거하는 기준을 찾지 못함
- '한국어'를 적용해보지 못함
- 같은 조건에서 모델을 학습했지만, 전혀 다른 결과가 나오는 이유를 찾지 못함



파란 선 : 단어 집합에서 희귀 단어의 비율

주황 선 : 전체 등장 빈도에서 희귀 단어 등장 빈도 비율