# 자연어처리 과제#5

1. 강의자료 11의 Naver 영화평 분류 프로그램을 NLTK 영화 분류와 유사하게 세 가지 방식으로 수행하도록 프로그램을 수정한다. 텍스트 전처리 부분은 강의자료의 프로그램으로 수행하고 세 가지 신경망 프로그램은 다음과 같이 수정한다. summary 함수를 이용하여 각 방식에서의 Parameter 수를 제시하고, 결과 그래프를 강의자료 20쪽의 그래프 형태로 제시하라.

**텍스트 전처리**

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

urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/mas
ter/ratings_train.txt", filename="ratings_train.txt")
urllib.request.urlretrieve("https://raw.githubusercontent.com/e9t/nsmc/mas
ter/ratings_test.txt", filename="ratings_test.txt")

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

| | id | document | label |
|---|---|---|---|
| 0 | 9976970 | 아 더빙.. 진짜 짜증나네요 목소리 | 0 |
| 1 | 3819312 | 흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나 | 1 |
| 2 | 10265843 | 너무재밓었다그래서보는것을추천한다 | 0 |
| 3 | 9045019 | 교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정 | 0 |
| 4 | 6483659 | 사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ... | 1 |

```
train_data.drop_duplicates(subset=['document'], inplace=True)

print('총 샘플의 수 :',len(train_data))
print(train_data.groupby('label').size().reset_index(name = 'count'))

train_data = train_data.dropna(how = 'any') # Null 값이 존재하는 행 제거
print(train_data.isnull().values.any()) # Null 값이 존재하는지 확인
```

```
# 한글과 공백을 제외하고 모두 제거
train_data['document'] = train_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣
]","", regex=True)
train_data[:5]

train_data = train_data.dropna(how = 'any')
print(len(train_data))

test_data.drop_duplicates(subset = ['document'], inplace=True) # document
열에서 중복인 내용이 있다면 중복 제거
test_data['document'] = test_data['document'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣
]","", regex=True) # 정규 표현식 수행
test_data['document'] = test_data['document'].str.replace('^ +', "",
regex=True) # 공백은 empty 값으로 변경
test_data['document'].replace('', np.nan, inplace=True) # 공백은 Null 값으로
변경
test_data = test_data.dropna(how='any') # Null 값 제거
print('전처리 후 테스트용 샘플의 개수 :',len(test_data))
```

```
총 샘플의 수 : 146183
   label  count
0      0  73342
1      1  72841
False
146182
전처리 후 테스트용 샘플의 개수 : 48852
```

```
stopwords = ['의','가','이','은','들','는','좀','잘','걍','과','도','를','으
로','자','에','와','한','하다']

okt = Okt()

X_train = []
for sentence in tqdm(train_data['document']):
    tokenized_sentence = okt.morphs(sentence, stem=True) # 토큰화
    stopwords_removed_sentence = [word for word in tokenized_sentence if
not word in stopwords] # 불용어 제거
    X_train.append(stopwords_removed_sentence)

X_test = []
for sentence in tqdm(test_data['document']):
    tokenized_sentence = okt.morphs(sentence, stem=True) # 토큰화
    stopwords_removed_sentence = [word for word in tokenized_sentence if
not word in stopwords] # 불용어 제거
    X_test.append(stopwords_removed_sentence)
```

```
100%|██████████████████████████████████| 146182/146182 [09:35<00:00,
253.84it/s]
100%|██████████████████████████████████| 48852/48852 [03:36<00:00,
225.41it/s]
```

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)

print(tokenizer.word_index)
```

```
{'영화': 1, '보다': 2, '을': 3, '없다': 4, '이다': 5, '있다': 6, '좋다': 7, '너무':
8, '다': 9, '정말': 10, '되다': 11, '재밌다': 12, ...(생략)..., '찎었': 43750, '디
케이드': 43751, '수간': 43752}
```

```
threshold = 3
total_cnt = len(tokenizer.word_index) # 단어의 수
rare_cnt = 0 # 등장 빈도수가 threshold보다 작은 단어의 개수를 카운트
total_freq = 0 # 훈련 데이터의 전체 단어 빈도수 총 합
rare_freq = 0 # 등장 빈도수가 threshold보다 작은 단어의 등장 빈도수의 총 합

# 단어와 빈도수의 쌍(pair)을 key와 value로 받는다.
for key, value in tokenizer.word_counts.items():
    total_freq = total_freq + value

    # 단어의 등장 빈도수가 threshold보다 작으면
    if(value < threshold):
        rare_cnt = rare_cnt + 1
        rare_freq = rare_freq + value

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

```
단어 집합(vocabulary)의 크기 : 43752
등장 빈도가 2번 이하인 희귀 단어의 수: 24337
단어 집합에서 희귀 단어의 비율: 55.62488571950996
전체 등장 빈도에서 희귀 단어 등장 빈도 비율: 1.8715872104872904
```

```
# 전체 단어 개수 중 빈도수 2이하인 단어는 제거.
# 0번 패딩 토큰을 고려하여 + 1
```

```
vocab_size = total_cnt - rare_cnt + 1
print('단어 집합의 크기 :',vocab_size)
```

```
단어 집합의 크기 : 19416
```

```
tokenizer = Tokenizer(vocab_size)
# 각 단어별로 정수를 배정
tokenizer.fit_on_texts(X_train)
# 배정된 정수를 이용하여 문자 텍스트를 숫자 시퀀스로 변환
X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

print(X_train[:3])
```

```
[[50, 454, 16, 260, 659], [933, 457, 41, 602, 1, 214, 1449, 24, 961, 675,
19], [386, 2444, 2315, 5671, 2, 222, 9]]
```

```
y_train = np.array(train_data['label'])
y_test = np.array(test_data['label'])

# 🚨 빈 샘플 제거 전에 패딩 적용을 먼저 해야됨! 🚨
# 시퀀스를 패딩하여 모두 동일한 길이로 만든다.
maxlen = 30
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)

drop_train = [index for index, sentence in enumerate(X_train) if
len(sentence) < 1]

# 빈 샘플들을 제거
X_train = np.delete(X_train, drop_train, axis=0)
y_train = np.delete(y_train, drop_train, axis=0)
print(len(X_train))
print(len(y_train))
```

```
146182
146182
```

## a) Flatten 방식

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

model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(Flatten())
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max',
verbose=1, save_best_only=True)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
['acc'])
history = model.fit(X_train, y_train, epochs=5, callbacks=[es, mc],
batch_size=64, validation_split=0.2)
```

```
Epoch 1/5
1826/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - acc: 0.7401 - loss: 0.5127
Epoch 1: val_acc improved from -inf to 0.80634, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 42s 22ms/step - acc: 0.7402 - loss: 0.5126
- val_acc: 0.8063 - val_loss: 0.4159
Epoch 2/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - acc: 0.8376 - loss: 0.3682
Epoch 2: val_acc improved from 0.80634 to 0.82522, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 40s 22ms/step - acc: 0.8376 - loss: 0.3682
- val_acc: 0.8252 - val_loss: 0.3914
Epoch 3/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - acc: 0.8543 - loss: 0.3371
Epoch 3: val_acc improved from 0.82522 to 0.83196, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 39s 22ms/step - acc: 0.8543 - loss: 0.3371
- val_acc: 0.8320 - val_loss: 0.3794
Epoch 4/5
1827/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - acc: 0.8732 - loss: 0.3079
Epoch 4: val_acc did not improve from 0.83196
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 39s 21ms/step - acc: 0.8732 - loss: 0.3079
- val_acc: 0.8297 - val_loss: 0.3909
Epoch 5/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step - acc: 0.8907 - loss: 0.2726
Epoch 5: val_acc did not improve from 0.83196
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 41s 22ms/step - acc: 0.8907 - loss: 0.2726
- val_acc: 0.8295 - val_loss: 0.4028
```

```
model.summary()
```

**Model: "sequential_4"**

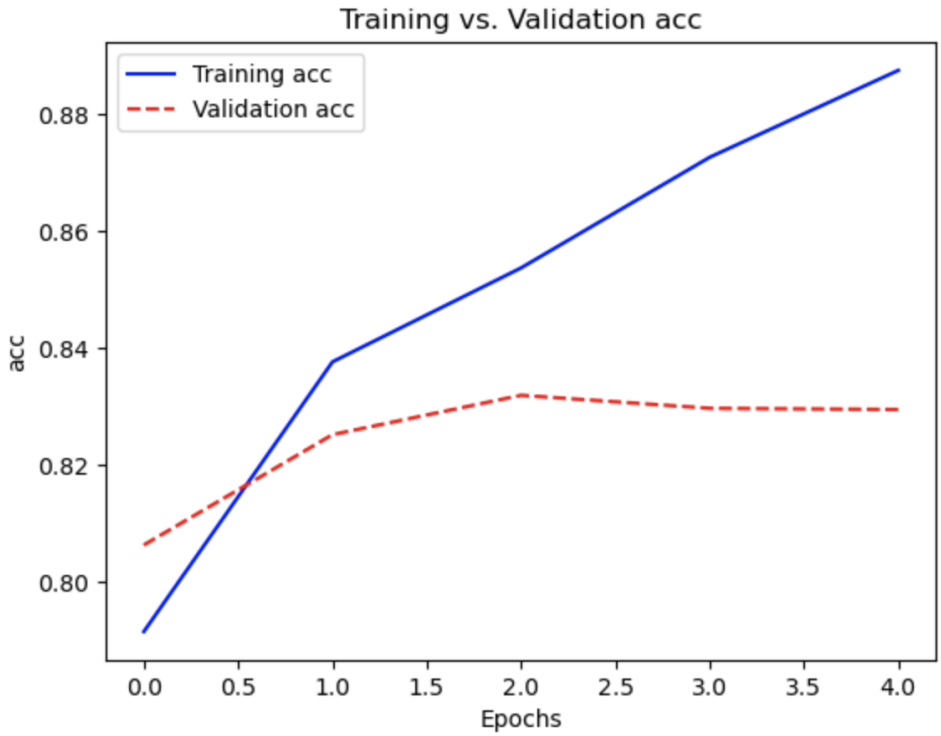
| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_4 (Embedding) | (None, 30, 100) | 1,941,600 |
| flatten_2 (Flatten) | (None, 3000) | 0 |
| dense_1 (Dense) | (None, 1) | 3,001 |

**Total params:** 3,889,204 (14.84 MB)
**Trainable params:** 1,944,601 (7.42 MB)
**Non-trainable params:** 0 (0.00 B)
**Optimizer params:** 1,944,603 (7.42 MB)

```
import matplotlib.pyplot as plt

def plot_results(history, metric):
    plt.plot(history.history[metric], 'b', label='Training '+metric)
    plt.plot(history.history['val_'+metric], 'r--', label='Validation
'+metric)
    plt.title('Training vs. Validation '+metric)
    plt.xlabel('Epochs')
    plt.ylabel(metric)
    plt.legend()
    plt.show()

plot_results(history, 'acc')
```



## b) RNN 방식

```
from tensorflow.keras.layers import SimpleRNN

model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(SimpleRNN(128))
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max',
verbose=1, save_best_only=True)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
['acc'])
history = model.fit(X_train, y_train, epochs=5, callbacks=[es, mc],
batch_size=64, validation_split=0.2)
```

```
Epoch 1/5
1827/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - acc: 0.7489 - loss: 0.4953
Epoch 1: val_acc improved from -inf to 0.81694, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 65s 35ms/step - acc: 0.7490 - loss: 0.4952
- val_acc: 0.8169 - val_loss: 0.4075
Epoch 2/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 32ms/step - acc: 0.8371 - loss: 0.3697
Epoch 2: val_acc did not improve from 0.81694
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 64s 35ms/step - acc: 0.8371 - loss: 0.3697
- val_acc: 0.7943 - val_loss: 0.4355
Epoch 3/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - acc: 0.8568 - loss: 0.3391
Epoch 3: val_acc improved from 0.81694 to 0.83487, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 68s 37ms/step - acc: 0.8568 - loss: 0.3391
- val_acc: 0.8349 - val_loss: 0.3916
Epoch 4/5
1827/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - acc: 0.8766 - loss: 0.2997
Epoch 4: val_acc did not improve from 0.83487
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 58s 32ms/step - acc: 0.8766 - loss: 0.2997
- val_acc: 0.8274 - val_loss: 0.3984
Epoch 5/5
1826/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - acc: 0.8923 - loss: 0.2693
Epoch 5: val_acc did not improve from 0.83487
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 59s 32ms/step - acc: 0.8923 - loss: 0.2693
- val_acc: 0.8176 - val_loss: 0.4369
```

```
model.summary()
```

**Model: "sequential_5"**

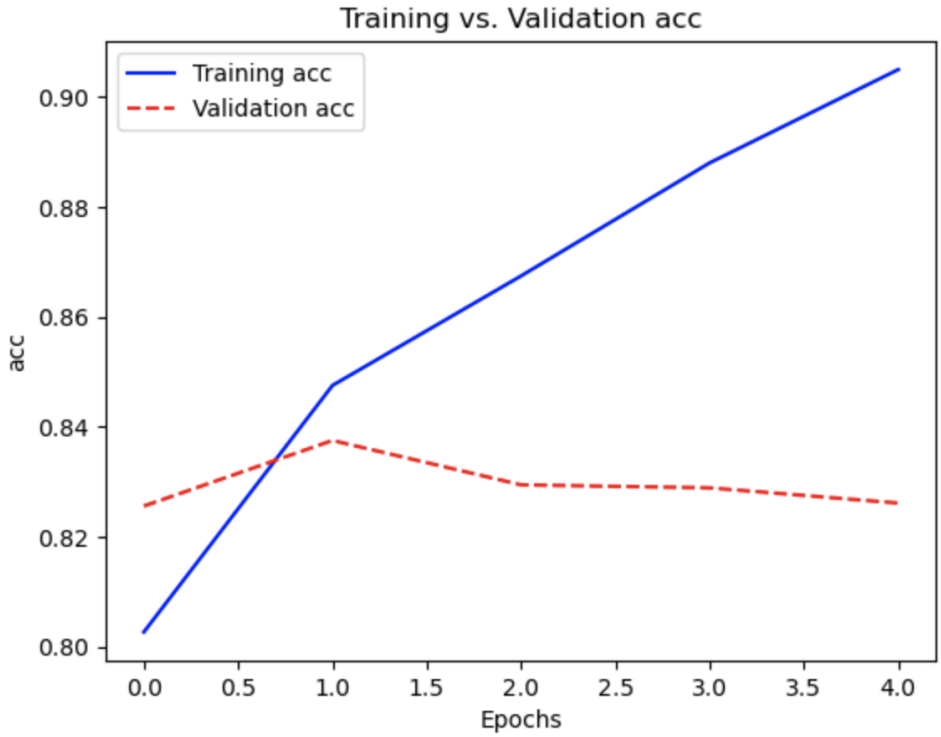
| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding_5 (Embedding) | (None, 30, 100) | 1,941,600 |
| simple_rnn (SimpleRNN) | (None, 64) | 10,560 |
| dense_2 (Dense) | (None, 1) | 65 |

**Total params:** 3,904,452 (14.89 MB)
**Trainable params:** 1,952,225 (7.45 MB)
**Non-trainable params:** 0 (0.00 B)
**Optimizer params:** 1,952,227 (7.45 MB)

```
import matplotlib.pyplot as plt

def plot_results(history, metric):
    plt.plot(history.history[metric], 'b', label='Training '+metric)
    plt.plot(history.history['val_'+metric], 'r--', label='Validation
'+metric)
    plt.title('Training vs. Validation '+metric)
    plt.xlabel('Epochs')
    plt.ylabel(metric)
    plt.legend()
    plt.show()

plot_results(history, 'acc')
```



## c) LSTM 방식

```
from tensorflow.keras.layers import LSTM

model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.keras', monitor='val_acc', mode='max',
verbose=1, save_best_only=True)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=
['acc'])
history = model.fit(X_train, y_train, epochs=5, callbacks=[es, mc],
batch_size=64, validation_split=0.2)
```

```
Epoch 1/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 95ms/step - acc: 0.7645 - loss: 0.4740
Epoch 1: val_acc improved from -inf to 0.83452, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 200s 108ms/step - acc: 0.7646 - loss:
0.4740 - val_acc: 0.8345 - val_loss: 0.3720
Epoch 2/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - acc: 0.8465 - loss: 0.3519
Epoch 2: val_acc improved from 0.83452 to 0.84205, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 201s 110ms/step - acc: 0.8465 - loss:
0.3519 - val_acc: 0.8420 - val_loss: 0.3548
Epoch 3/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step - acc: 0.8631 - loss: 0.3179
Epoch 3: val_acc improved from 0.84205 to 0.84930, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 223s 122ms/step - acc: 0.8631 - loss:
0.3179 - val_acc: 0.8493 - val_loss: 0.3438
Epoch 4/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 114ms/step - acc: 0.8774 - loss: 0.2958
Epoch 4: val_acc improved from 0.84930 to 0.85296, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 230s 126ms/step - acc: 0.8774 - loss:
0.2958 - val_acc: 0.8530 - val_loss: 0.3383
Epoch 5/5
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 0s 108ms/step - acc: 0.8868 - loss: 0.2750
Epoch 5: val_acc improved from 0.85296 to 0.85467, saving model to
best_model.keras
1828/1828 ━━━━━━━━━━━━━━━━━━━━ 217s 119ms/step - acc: 0.8868 - loss:
0.2750 - val_acc: 0.8547 - val_loss: 0.3347
```

```
model.summary()
```

**Model: "sequential"**

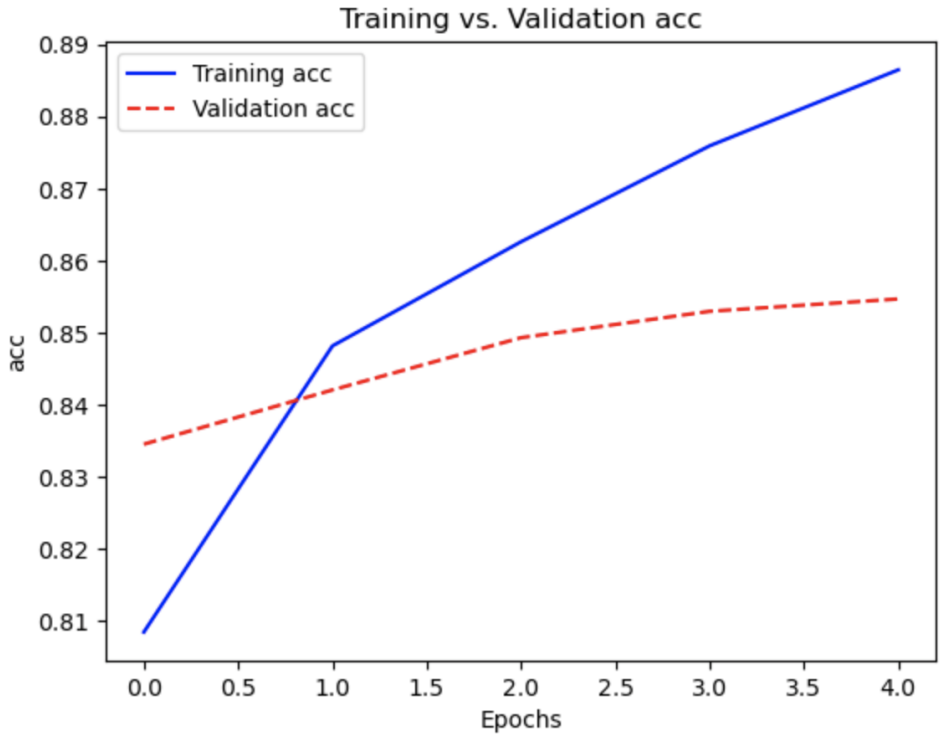
| Layer (type) | Output Shape | Param # |
|---|---|---|
| embedding (Embedding) | (None, 30, 100) | 1,941,600 |
| lstm (LSTM) | (None, 128) | 117,248 |
| dense (Dense) | (None, 1) | 129 |

**Total params:** 4,117,956 (15.71 MB)
**Trainable params:** 2,058,977 (7.85 MB)
**Non-trainable params:** 0 (0.00 B)
**Optimizer params:** 2,058,979 (7.85 MB)

```
import matplotlib.pyplot as plt

def plot_results(history, metric):
    plt.plot(history.history[metric], 'b', label='Training '+metric)
    plt.plot(history.history['val_'+metric], 'r--', label='Validation
'+metric)
    plt.title('Training vs. Validation '+metric)
    plt.xlabel('Epochs')
    plt.ylabel(metric)
    plt.legend()
    plt.show()

plot_results(history, 'acc')
```



---

## d) 결과 분석

**파라미터 수**는 Flatten 방식이 3,889,204개로 가장 적고, RNN 방식이 3,904,452개로 그다음, LSTM 방식이 4,117,956개로 가장 많다.

**결과의 정확도**는 LSTM 방식이 가장 높은 성능을 보이며, 이전 에포크보다 정확도가 낮아지는 현상이 없는 것을 보니 과잉적합이 발생하지 않은것을 확인할 수 있다. 그 다음으로 높은 성능을 보이는 것은 Flatten 방식인데, 2에포크부터 과잉적합이 발생해 정확도가 낮아지는 것을 확인할 수 있다. 가장 낮은 성능을 보이는 것은 RNN 방식이고, 1에포크부터 과잉적합이 발생해 정확도가 낮아지며 Flatten방식보다 더 성능이 떨어지는 것을 확인할 수 있다.