과제 정의

- **1. Text Preprocessing - Text to Word Index ID**
  - #Write your code 파트 code 완성하기
  - 1.1 한글 단어 Token화 하기
    - 빈도를 기준으로 상위 10,000개의 단어들만 선택
    - Counter 함수 이용
  - 1.2
    - 각 단어에 대해서 index 생성하기 (실습 데이터 처럼 단어를 id index로 표현하기 위함)
    - 각 index에 대해서 단어 기억하기
    - 각 문서를 상위 10000개 단어들에 대해서 index 번호로 표현하기

과제

Write your code 채워놓은 부분

```
#Dictionary Function
# 각 단어에 대해서 index 생성하기
words_dic = {word: index+3 for index, word in enumerate(common_words)}
#Write your code

# 각 index에 대해서 단어 기억하기
#Write your code
print(words_dic["영화"])
print(words_dic["너무"])
```

```
# 각 문서를 상위 10000개 단어들에 대해서 index 번호로 표현하기
# 문장의 시작 토큰을 1로, 어휘 사전에 없는 단어는 2로 설정해두었
filtered_indexed_words = []
#Write your code
for review_words in words_list:
    indexed_words = [1]
    for word in review_words:
        if word in words_dic:
            indexed_words.append(words_dic[word])
        else:
            indexed_words.append(2)
    filtered_indexed_words.append(indexed_words)
```

+ Code + Markdown

```
print(filtered_indexed_words[104])
```

```
[1, 379, 672, 1768, 1001, 5, 3, 256, 2]
```

+ Code + Markdown

가장 빈도 높은 10000개 단어들을 기준으로 words_dic 생성(단어를 key값으로, 인덱스 값을 value값으로 하는 딕셔너리)

이때 문장의 시작을 나타내는 토큰(1), 상위 10000개 단어 어휘사전에 포함되지 않아서 인덱스를 찾을 수 없는 단어를 위한 토큰(2)을 위해서 실제 단어 인덱스들은 3부터 시작하도록 했고
filtered_indexed_words에는 토큰화된 리뷰들(words_list)을 순회하며 단어->인덱스로 맵핑하여 다시 저장함.

과제 정의

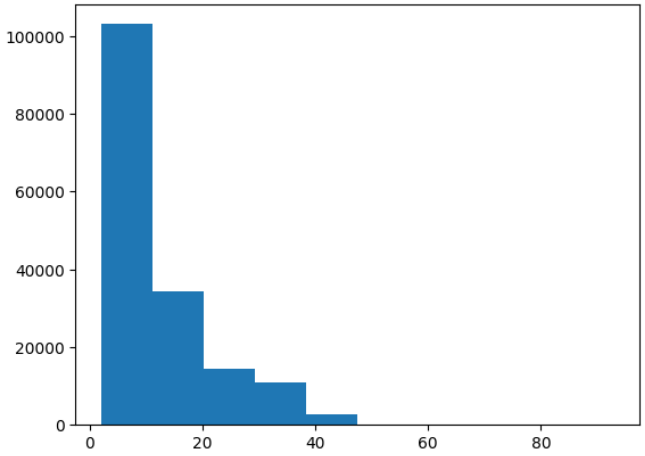
- **2. Dataset Preparation for CNN models**
  - #Write your code 파트 code 완성하기
  - 2.1 X - input data 처리 (text tokens id_index to padded X)
  - 2.2 y - label data 처리 (one_hot_encoded y)
  - 2.3 Train / Test Split
    - Test size – 0.1

과제 정의

2.1 X - input data 처리

```
from tensorflow.keras.preprocessing import sequence
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt

#Write your code
# 2.1 X - input data 처리 (text tokens id_index to padded X) #max length ->padding
'''max_len = 0
for x in filtered_indexed_words:
    if len(x) > max_len:
        max_len = len(x)
print(max_len)''' #결과는 max_len 93이었다
#lengths = np.array([len(x) for x in filtered_indexed_words])
#print(np.mean(lengths), np.median(lengths)) #결과는 각각 12.4, 9.0이었다
#plt.hist(lengths)
#plt.show()
#그래프 그려 본 결과 대부분의 리뷰 길이는 20 이하임.
max_len = 20
input_data = sequence.pad_sequences(filtered_indexed_words, maxlen=max_len)
```



22]:

```
print(input_data[0])
```

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   1 206 251 123   4
 347   5]
```

적절한 max_len을 찾기 위해 리뷰들의 길이 평균, 중앙값을 찾고 길이의 분포 정도를 matplotlib을 통해 시각화 해 본 결과가 다음과 같다. Max_len을 결국 20으로 설정했고 리뷰가 20보다 짧은 경우 앞 부분에 패딩값인 0을, 긴 경우 앞 부분을 자르는 방식으로 input data를 전처리 했다

과제 정의

2.2 Y - label data 처리

```
# 2.2 y - label data 처리 (one_hot_encoded y) #위에 있는 코드 가져다 작성하면 됨
label_data = to_categorical(labels)
```

One-hot encoding 방식을 사용해 레이블 값(긍정/부정)이 2차원으로 표현될 수 있도록 전처리 했다.

```
print(label_data)
```

```
[[1. 0.]
 [0. 1.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

2.3 Train & Test set Split

```
from sklearn.model_selection import train_test_split
#Write your code
# 2.3 Train / Test Split
train_input, test_input, train_label, test_label = train_test_split(input_data, label_data, test_size = 0.1, random_state = 42)
print(train_input.shape, train_label.shape, test_input.shape, test_label.shape)
```

```
(148845, 20) (148845, 2) (16539, 20) (16539, 2)
```

사이킷런의 `train_test_split` 함수를 이용해 Train, test 셋의 비율을 9:1로 나누었고(매개변수 test_size=0.1) Shape 확인을 통해 잘 split 되었음을 확인

과제 정의

- **3. Model Build & Setting**
  - #Write your code 파트 code 완성하기
  - Model Build

```
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_shape=(max_len,)))
model.add(layers.Conv1D(32, 7, strides=1, activation='relu'))
model.add(layers.MaxPool1D(5))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))
model.summary()
```

  - Setting
  - Early Stopping 구현
  - model.compile을 이용하여 optimizer(원하는 optimizer 사용), loss (binary_crossentropy), metrics (accuracy) 구현
  - 4주차-2 Deep Learning Recap 2 파트 참고

- **4. Model Training**
  - history = model.fit(x_train, y_train_one_hot, epochs=3, batch_size=128, validation_split=0.2, callbacks=[es])
  - 위 코드를 참고하여 본인만의 모델을 만들어 loss 최소화, accuracy 최대화

과제 정의

- **실험 1**

```
#1차 실험

from tensorflow.keras import layers
from tensorflow.keras import models

model = models.Sequential()

#Write your code - model build
model.add(layers.Embedding(len(words_dic)+1, 128, input_shape=(max_len,)))
model.add(layers.Conv1D(64, 5, activation='relu'))
model.add(layers.MaxPool1D(3))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))

model.summary()
```

```
Model: "sequential_3"
```

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 20, 128)	1,280,128
conv1d_3 (Conv1D)	(None, 16, 64)	41,024
max_pooling1d_3 (MaxPooling1D)	(None, 5, 64)	0
flatten_3 (Flatten)	(None, 320)	0
dense_6 (Dense)	(None, 32)	10,272
dense_7 (Dense)	(None, 2)	66

```
Total params: 1,331,490 (5.08 MB)
Trainable params: 1,331,490 (5.08 MB)
Non-trainable params: 0 (0.00 B)
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop # you can add more optimizers

#Write your code - model setting
#Early Stopping 구현
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 2)
#optimizer & loss
model.compile(optimizer = 'adam', loss='binary_crossentropy', metrics = ['accuracy'])
```

실험1 모델 구조

1) Embedding Layer: 128차원의 단어 embedding 생성
2) Conv1D: size가 5인 64개의 filter 사용하는 Convolution 레이어
3) MaxPool1D: window size를 5로 하는 max-pooling 레이어
4) Flatten: 벡터를 평면화
5) Dense: relu 활성화함수 사용하는 Fully connected layer
6) Dense: softmax 활성화함수(2클래스 분류) 사용하는 Fully connected layer

Early stopping 및 모델 컴파일

Val_loss 값이 더 이상 줄어들지 않을 때 early stopping 하도록 구현, patience값은 2. optimizer는 Adam 사용

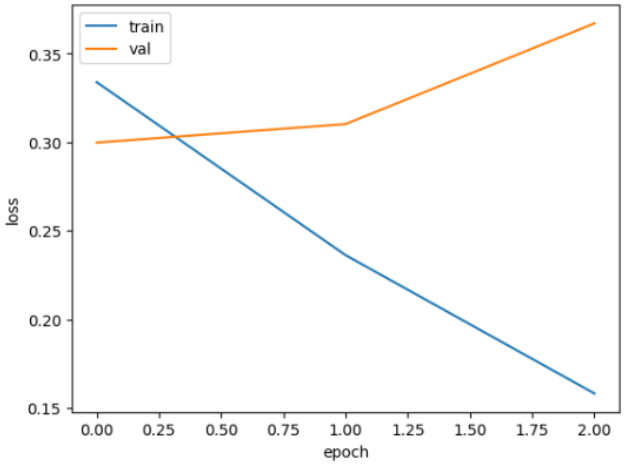
과제 정의

- **실험 1**

```
#Write your code - model training
history = model.fit(train_input, train_label, epochs = 5, batch_size = 64, validation_split=0.2, callbacks=[es])
```

```
Epoch 1/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 28s 14ms/step - accuracy: 0.8053 - loss: 0.4018 - val_accuracy: 0.8710 - val_loss: 0.2997
Epoch 2/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 40s 14ms/step - accuracy: 0.9056 - loss: 0.2304 - val_accuracy: 0.8702 - val_loss: 0.3102
Epoch 3/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 27s 14ms/step - accuracy: 0.9407 - loss: 0.1521 - val_accuracy: 0.8622 - val_loss: 0.3670
Epoch 3: early stopping
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train','val'])
plt.show()
```




```
test_loss, test_acc = model.evaluate(test_input,test_label)
```

```
517/517 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8633 - loss: 0.3597
```

```
print("Loss:", test_loss)
print("Accuracy:", test_acc)
```

```
Loss: 0.3499686121940613
Accuracy: 0.8671624660491943
```

+ Code + Markdown

실험1 학습
에폭 수는 5, batch size는 64, validation set은 0.2로 설정하고 학습

결과 및 분석
Early stopping에 의해 3에폭을 돌지 못하고 학습 중단되었다.
Test set에서의 Accuracy값은 **0.867**
학습 그래프를 확인했을 때 train set의 loss값은 꾸준히 감소했으나 validation set에서 1에폭 이후로 loss값이 다시 증가함을 보임.
이는 training set에 대한 overfitting이 발생했음을 추정할 수 있다.

과제

과제 정의

- **실험 2**

[36]:

```
#2차 실험

from tensorflow.keras import layers
from tensorflow.keras import models

model = models.Sequential()

#Write your code - model build
model.add(layers.Embedding(len(words_dic)+1, 64, input_shape=(max_len,)))
model.add(layers.Conv1D(16, 5, activation='relu'))
model.add(layers.MaxPool1D(3))
model.add(layers.Flatten())
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(2, activation='softmax'))

model.summary()
```

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop # you can add more optimizers

#Write your code - model setting
#Early Stopping 구현
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 2)
#optimizer & loss
model.compile(optimizer = 'adam', loss='binary_crossentropy', metrics = ['accuracy'])
```

```
Model: "sequential_5"
```

Layer (type)	Output Shape	Param #
embedding_5 (Embedding)	(None, 20, 64)	640,064
conv1d_5 (Conv1D)	(None, 16, 16)	5,136
max_pooling1d_5 (MaxPooling1D)	(None, 5, 16)	0
flatten_5 (Flatten)	(None, 80)	0
dense_10 (Dense)	(None, 16)	1,296
dense_11 (Dense)	(None, 2)	34

```
Total params: 646,530 (2.47 MB)
Trainable params: 646,530 (2.47 MB)
Non-trainable params: 0 (0.00 B)
```

실험2 모델 구조
실험 2에서는 실험 1에서 발생했던 overfitting 문제를 개선하기 위해 model complexity를 줄이고자 하였다.

1) Embedding Layer: 64차원의 단어 embedding 생성(128->64)
2) Conv1D: size가 5인 16개의 filter 사용하는 Convolution 레이어(64->16)
3) MaxPool1D: window size를 5로 하는 max-pooling 레이어
4) Flatten: 벡터를 평면화
5) Dense: relu 활성화함수 사용하고 unit 16개로 이루어진 Fully connected layer(unit 개수를 32->16개)
6) Dense: softmax 활성화함수(2클래스 분류) 사용하는 Fully connected layer

Early stopping 및 모델 컴파일
Val_loss 값이 더 이상 줄어들지 않을 때 early stopping 하도록 구현, patience값은 2. optimizer는 adam 사용

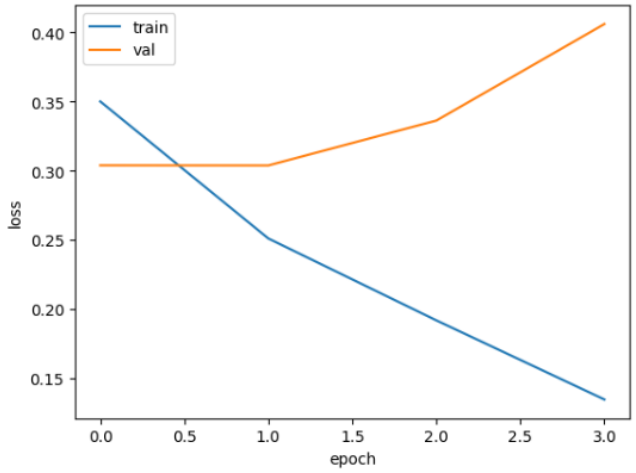
과제 정의

- **실험 2**

```
#Write your code - model training
history = model.fit(train_input, train_label, epochs = 5, batch_size = 64, validation_split=0.2, callbacks=[es])
```

```
Epoch 1/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.7771 - loss: 0.4348 - val_accuracy: 0.8691 - val_loss: 0.3039
Epoch 2/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - accuracy: 0.8971 - loss: 0.2467 - val_accuracy: 0.8717 - val_loss: 0.3038
Epoch 3/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - accuracy: 0.9268 - loss: 0.1862 - val_accuracy: 0.8659 - val_loss: 0.3363
Epoch 4/5
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - accuracy: 0.9523 - loss: 0.1292 - val_accuracy: 0.8591 - val_loss: 0.4061
Epoch 4: early stopping
```

```
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train','val'])
plt.show()
```




```
test_loss, test_acc = model.evaluate(test_input,test_label)
```

```
517/517 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8625 - loss: 0.3978
```

```
print("Loss:", test_loss)
print("Accuracy:", test_acc)
```

```
Loss: 0.39203962683677673
Accuracy: 0.8650462627410889
```

실험2 학습
실험1의 조건과 똑같이 학습진행

결과 및 분석
Early stopping에 의해 4에폭까지 돌고 학습 중단되었다.
Test set에서의 Accuracy값은 **0.865**
실험 1에서와 마찬가지로 validation 셋의 loss가 증가함을 보았을 때 이번에도 Overfitting이 발생했음을 확인할 수 있다.

과제

과제 정의

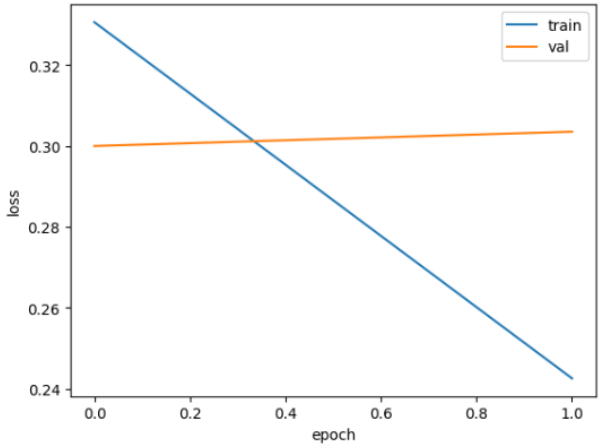
- **실험 3**

```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop # you can add more optimizers

#Write your code - model setting
#Early Stopping 구현
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1)
#optimizer & loss
model.compile(optimizer = 'adam', loss='binary_crossentropy', metrics = ['accuracy'])
```

```
#Write your code - model training
history = model.fit(train_input, train_label, epochs = 3, batch_size = 16, validation_split=0.2, callbacks=[es])
```

```
Epoch 1/3
7443/7443 ━━━━━━━━━━━━━━━━━━━━ 46s 6ms/step - accuracy: 0.8158 - loss: 0.3887 - val_accuracy: 0.8711 - val_loss: 0.2999
Epoch 2/3
7443/7443 ━━━━━━━━━━━━━━━━━━━━ 47s 6ms/step - accuracy: 0.9011 - loss: 0.2373 - val_accuracy: 0.8722 - val_loss: 0.3035
Epoch 2: early stopping
```




```
print("Loss:", test_loss)
print("Accuracy:", test_acc)
```

```
Loss: 0.2917397618293762
Accuracy: 0.8770179748535156
```

+ Code + Markdown

실험3
모델 구조는 실험 2와 동일하게 사용하고
Earlystopping 에서 patience 값을 디폴트(0)으로 변경
학습 조건 변경 내역
Batch size를 64에서 16으로 감소시켰다.
5 이상의 Epoch 은 큰 의미가 없는 것 같아 3에폭으로 감소시켰다.

결과 및 분석
Early stopping에 의해 2에폭까지 돌고 학습 중단되었다.
Test set에서의 Accuracy값은 **0.877**
학습과정에서 validation set의 loss값 증가추세가 실험 1, 실험 2에 비해 완화되어 과적합 정도가 소폭 감소했다고 해석 가능하다.

과제 정의

- **실험 4**

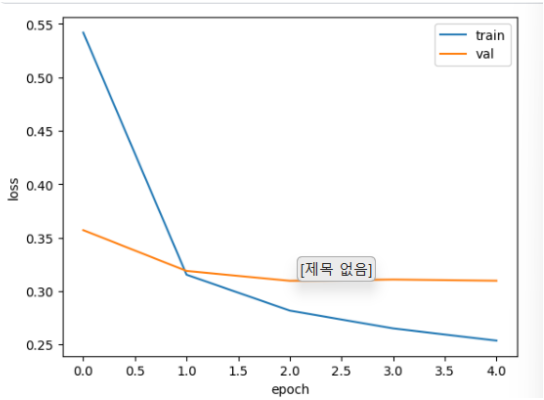
```
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.optimizers import RMSprop # you can add more optimizers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.optimizers import Adagrad

#Write your code - model setting
#Early Stopping 구현
es = EarlyStopping(monitor = 'val_loss', mode = 'min', verbose = 1, patience = 2)
#optimizer & loss

model.compile(optimizer = Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics = ['accuracy'])
```

```
#Write your code - model training
history = model.fit(train_input, train_label, epochs = 10, batch_size = 64, validation_split=0.2, callbacks=[es])
```

```
Epoch 1/10
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 15s 7ms/step - accuracy: 0.6139 - loss: 0.6429 - val_accuracy: 0.8402 - val_loss: 0.3570
Epoch 2/10
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 20s 7ms/step - accuracy: 0.8634 - loss: 0.3228 - val_accuracy: 0.8606 - val_loss: 0.3188
Epoch 3/10
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 15s 8ms/step - accuracy: 0.8820 - loss: 0.2830 - val_accuracy: 0.8666 - val_loss: 0.3095
Epoch 4/10
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 14s 8ms/step - accuracy: 0.8921 - loss: 0.2599 - val_accuracy: 0.8678 - val_loss: 0.3107
Epoch 5/10
1861/1861 ━━━━━━━━━━━━━━━━━━━━ 14s 7ms/step - accuracy: 0.8976 - loss: 0.2516 - val_accuracy: 0.8703 - val_loss: 0.3096
Epoch 5: early stopping
```



실험4
모델 구조는 실험 2와 동일하게 사용하고
Adam optimizer의 learning rate을 0.0001로 더 작게 초기화
Early stopping에서 patience 값을 2로 다시 변경
에폭 수를 10으로 늘리고 batch_size는 다시 64로 변경

결과 및 분석
Early stopping에 의해 5에폭 도중 멈춤
학습과정에서 에폭 진행에 따라 validation set의 loss값도 training set과 마찬가지로 전반적 감소 추세를 보임

과제 정의

- **5. Test Model**
  - test_loss, test_acc = model.evaluate(X_test,y_test)
  - print("Loss:", test_loss)
  - print("Accuracy:", test_acc)

5. Test Model

[143]:
```
test_loss, test_acc = model.evaluate(test_input,test_label)
```

```
517/517 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.8681 - loss: 0.3072
```

```
print("Loss:", test_loss)
print("Accuracy:", test_acc)
```

```
Loss: 0.29857712984085083
Accuracy: 0.8716367483139038
```

+ Code + Markdown

실험4 모델의 테스트 결과로
테스트 셋 상의 정확도는 0.872 , 손실값(binary cross entropy)은 0.299가 나왔다.

과제 정의

- **6. Model Inference**
  - #Write your code 파트 code 완성하기
  - 테스트셋에서 샘플 하나를 뽑고, 테스트셋의 X 데이터 토큰을 (id_index) 다시 자연어로 출력 하는 코드 작성
  - 테스트셋의 샘플 하나를 모델에 넣어서 모델 예측값과 실제 정답 값을 비교하는 코드 작성

```
#Write your code - model inference #Test-set에서 하나를 뽑아서 index-->word
idx_words_dic = {index:word for word, index in words_dic.items()}
import random
for i in range(5):
    ri = random.randint(0, len(test_input)-1)
    sample = test_input[ri]
    words = list(map(lambda token: idx_words_dic.get(token) if token not in [0,1,2] else None, sample))
    words = [word for word in words if word is not None]
    print("real review: ", words)
    print("answer: ", test_label[ri])
    pred = model.predict(np.array(sample).reshape(1,-1))
    print(pred)
    if pred[0][0] > pred[0][1] and test_label[ri][0] == 1:
        print("CORRECT")
    elif pred[0][0] < pred[0][1] and test_label[ri][1] == 1:
        print("CORRECT")
    else:
        print("INCORRECT")
```

1) 기존의 word(key)-index(value)로 저장된 words_dic을 이용해 key와 value값을 바꾼 inverted dictionary인 idx_words_dic 생성
2) 랜덤하게 샘플 5개를 테스트셋에서 뽑기
3) 인풋 샘플에서 토큰이 0(패딩)/1(시작토큰)/2(BOG에 없는 단어)일 경우 무시하고 유효한 토큰만 딕셔너리를 사용해 자연어 단어로 바꿔주기
4) 그렇게 자연어로 바꾼 Real revie와 정답 라벨값을 출력하고
5) 모델이 예측하는 긍정/부정 결과(이는 각 softmax에 의해 확률로 나온다)를 출력한다
6) 긍정 확률과 부정 확률 크기 비교를 통해 더 큰 값을 제대로 맞춘 경우 CORRECT 그렇지 않은 경우 INCORRECT 출력

과제 정의

- **6. Model Inference**
  - #Write your code 파트 code 완성하기
  - 테스트셋에서 샘플 하나를 뽑고, 테스트셋의 X 데이터 토큰을 (id_index) 다시 자연어로 출력 하는 코드 작성
  - 테스트셋의 샘플 하나를 모델에 넣어서 모델 예측값과 실제 정답 값을 비교하는 코드 작성

출력결과 모두 부정적인 리뷰를 부정적인 리뷰라고 잘 예측하는 모습을 볼 수 있다.

```
real review:  ['뭐', '하나', '갖춘', '없네', '재미', '교훈']
answer:  [1. 0.]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 26ms/step
[[0.9240606  0.07593945]]
CORRECT
real review:  ['받고', '봐', '야할', '수준', '영화관', '나오면', '관객', '모두', '짜증', '내는', '영화니', '시간', '많아', '할', '일', '없', '돈', '버린', '생각하고', '보세']
answer:  [1. 0.]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
[[9.9992347e-01 7.6506534e-05]]
CORRECT
real review:  ['기대', '별로', '재미', '없었', '거', '탄', '나오면', '봐']
answer:  [1. 0.]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
[[0.9940069  0.00599306]]
CORRECT
real review:  ['평론가', '평점', '점', '이', '영화', '좋아하는', '그룹', '대', '여자', '입니', '이', '가지', '이', '영화', '모든', '것', '평가', '할', '수', '있다']
answer:  [1. 0.]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
[[0.99438465 0.00561539]]
CORRECT
real review:  ['너무', '높아', '내려', '봄', '전혀', '이입', '되', '않은', '개연', '없는', '전개', '현실', '결여', '감독', '믿고', '봤', '진짜', '저', '별로', '였']
answer:  [1. 0.]
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
[[9.9945587e-01 5.4410094e-04]]
CORRECT
```