6장. 순환신경망

학습목표

- 컨볼루션 신경망과 함께 양대산맥을 이루는 순환 신경망(RNN; Recurrent Neural Network)
  - 음성 인식, 기계 번역(네이버 파파고, 구글 번역기)
  - 주식, 온도, 매출과 같이 시간이 지남에 따라 변화하는 데이터를 활용하여 주식 종가, 날씨, 상점 매출 예측
  - 자연어 처리(NLP; Natural Language Preprocessing)

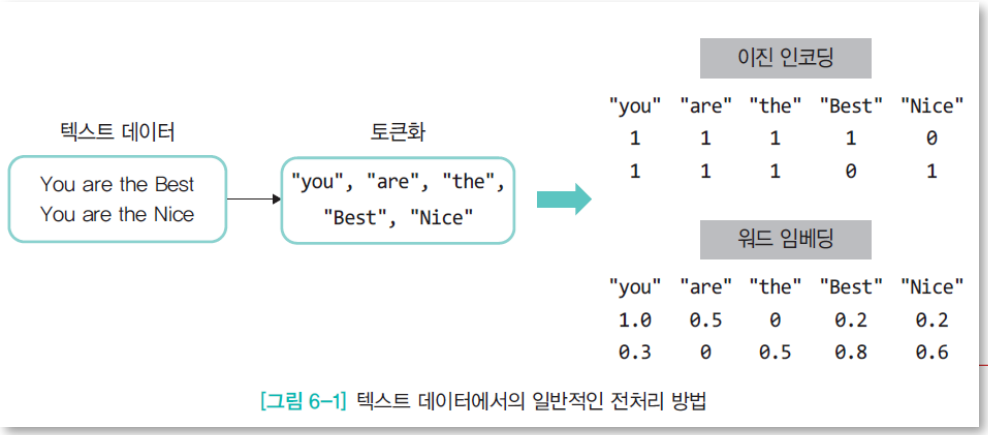
- 이번 장에서는 다음 내용을 다뤄봅니다.
  - Embedding, SimpleRNN, LSTM
  - IMDB 데이터셋 (영화사이트 리뷰 데이터)

Embedding – 원리 이해하기

- Embedding층은 **수많은 단어(또는 데이터)를 벡터 형태로 표현**할 수 있어 텍스트 분류를 위해 사용하는 가장 기본에 해당하는 층
  - 사전 학습된 가중치를 불러와 사용할 수 있음(실습을 참고)
  - 토큰, 토큰화, 텍스트 분류에 사용되는 기본적 용어를 알고 사용해보자

- **토큰(Token)**
  - 문법적으로 더 이상 나눌 수 없는 언어 요소
  - 이를 수행하는 작업을 **토큰화(Tokenizer)**라고 함

- 텍스트 데이터를 신경망에 입력하기 위해서 일반적으로 토큰화 작업을 수행하고 정의된 토큰에 고유 인덱스를 부여한 뒤, 인코딩을 통해 적절한 형태로 바꿔주는 전처리 작업 과정을 거치게 됨
  - 원-핫 인코딩, 이진 인코딩
  - 워드 임베딩(Word Embedding)



[그림 6-1] 텍스트 데이터에서의 일반적인 전처리 방법

Embedding – 원리 이해하기

- 토큰화 작업을 수행해보자
  - tensorflow.keras.preprocessing.text 모듈에서 이를 위한 함수를 제공

- 데이터 준비

```
04  texts = ['You are the Best',
05           'You are the Nice']
```

- Tokenizer() 함수를 사용하여 토큰화 작업 수행
  - oov_token: 데이터에 나타나지 않은 단어를 전달된 단어로 교체
  - fit_on_texts() 함수를 통해 데이터에 적용하고, texts_to_sequences() 함수로 변환

```
07  tokenizer = Tokenizer(num_words = 10, oov_token = '<OOV>')
08  tokenizer.fit_on_texts(texts)
09
10  # 텍스트 데이터를 정수 인덱스 형태로 변환합니다.
11  sequences = tokenizer.texts_to_sequences(texts)
```

```
{'<OOV>': 1, 'you': 2, 'are': 3, 'the': 4, 'best': 5, 'nice': 6}
```

Embedding – 원리 이해하기

- 새로운 데이터 **“You are the One”**
  - ‘One’ 단어는 새로 등장했기 때문에, ‘<OOV>’로 대체됨

```
test sequences: [[2, 3, 4, 1]]
```

  - ‘You’, ‘are’, ‘the’는 각각 2, 3, 4로 변환

- sequences_to_matrix() 함수를 사용하면 이진 형태로 인코딩된 결과를 얻을 수 있음

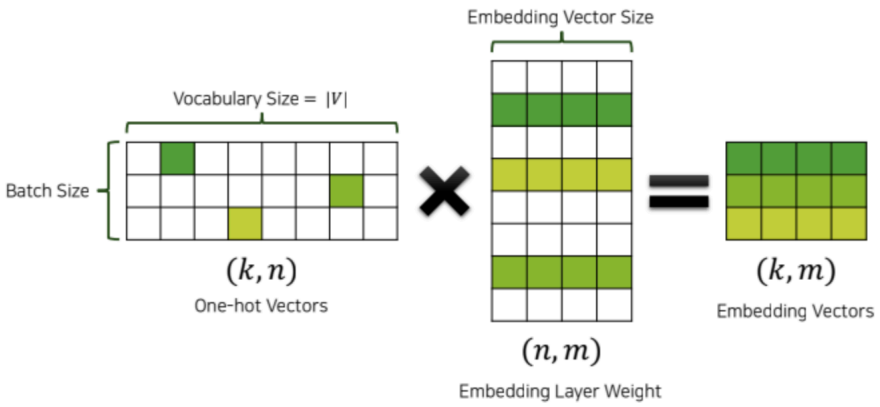
```
binary_vectors:
 [[0. 0. 1. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 0. 1. 0. 0. 0.]]
```

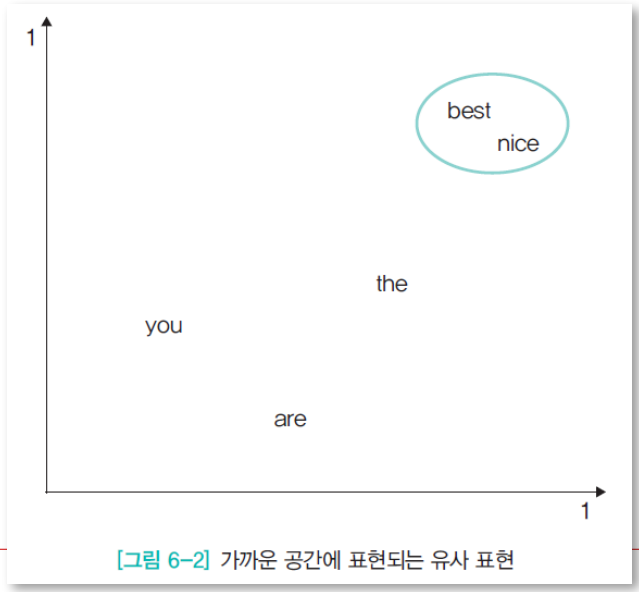
- sequences_to_matrix() 또는 to_categorical() 함수에서 얻을 수 있는 결과를 **희소 행렬(Sparse Matrix)**라고 표현하며, 이와 반대되는 개념을 **밀집 행렬(Dense Matrix)**라고 표현함

Embedding – 원리 이해하기

- 희소행렬
  - **존재하는 단어의 인덱스를 제외하고 전부 0으로 표현**
  - 고차원이며, 단어의 유사성(Similarity)를 표현할 수 없음
  - 행렬의 고차원으로 인해 불필요한 계산이 추가되며, 차원의 저주(Curse of Dimensionality)를 야기함

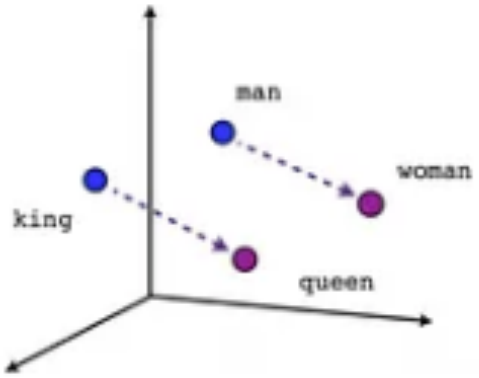
- 밀집행렬
  - **각 단어의 관계를 실수로 표현**하며, 저차원에 해당
  - 행렬에 속해있는 실숫값은 0과 1로 직접 지정해주는 희소행렬과 다르게 데이터를 기반으로 조정
  - 유사한 의미를 가지는 단어는 비슷한 공간에 표현(매핑)될 것



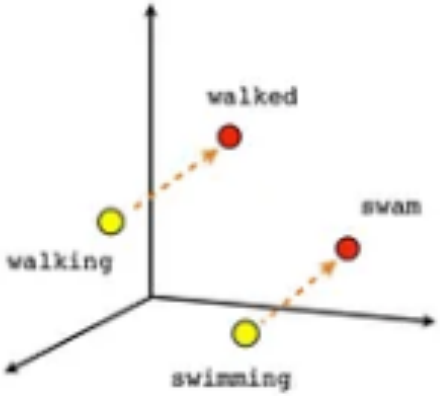


[그림 6-2] 가까운 공간에 표현되는 유사 표현

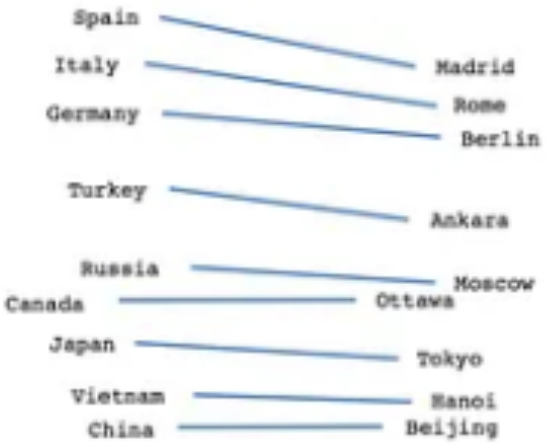
Embedding – 원리 이해하기



Male-Female



Verb tense



Country-Capital

Embedding – 실습

```
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical

texts = ['You are the Best',
         'You are the Nice']

tokenizer = Tokenizer(num_words = 10, oov_token = '<OOV>')
tokenizer.fit_on_texts(texts)

print(tokenizer.word_index)
```

```
{'<OOV>': 1, 'you': 2, 'are': 3, 'the': 4, 'best': 5, 'nice': 6}
```

Embedding – 실습

```
# 텍스트 데이터를 정수 인덱스 형태로 변환합니다.
sequences = tokenizer.texts_to_sequences(texts)

# 이진 형태로 인코딩 합니다.
binary_results = tokenizer.sequences_to_matrix(sequences, mode = 'binary')

print('--------------------')

print(f'sequences: {sequences}\n')
print(f'binary_vectors:\n {binary_results}\n')
```

```
--------------------
sequences: [[2, 3, 4, 5], [2, 3, 4, 6]]

binary_vectors:
 [[0. 0. 1. 1. 1. 1. 0. 0. 0. 0.]
 [0. 0. 1. 1. 1. 0. 1. 0. 0. 0.]]
```

Embedding – 실습

```
test_text = ['You are the One']
test_seq = tokenizer.texts_to_sequences(test_text)

print(f'test sequences: {test_seq}')
```

```
test sequences: [[2, 3, 4, 1]]
```

Embedding – 실습

Gensim은 최신 통계 기계 학습을 사용하여 비지도 주제 모델링, 문서 색인화, 유사성 검색 및 기타 자연어 처리 기능을 위한 오픈 소스 라이브러리 입니다.

```
!pip install gensim
```

Embedding – 실습

```
import gensim.downloader as api
for model_name, model_data in sorted(api.info()['models'].items()):
    print(
        '%s (%d records): %s' % (
            model_name,
            model_data.get('num_records', -1),
            model_data['description'][:40] + '...',
        )
    )
```

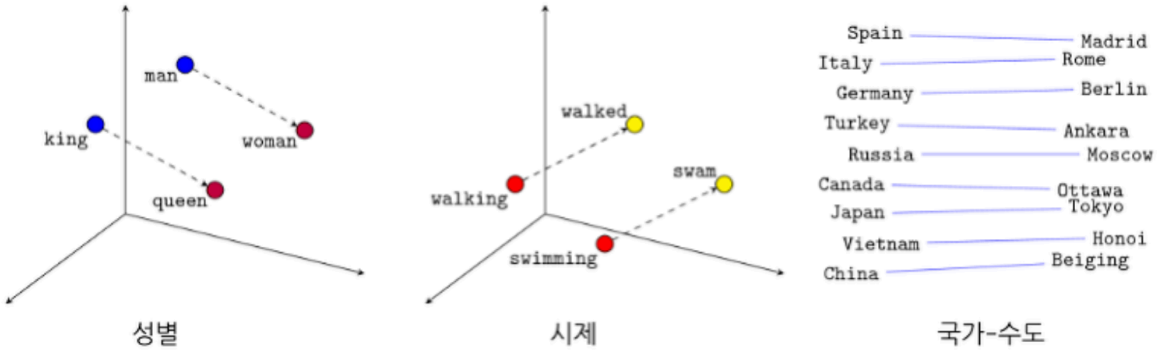
```
__testing_word2vec-matrix-synopsis (-1 records): [THIS IS ONLY FOR TESTING] Word vecrors ...
conceptnet-numberbatch-17-06-300 (1917247 records): ConceptNet Numberbatch consists of state...
fasttext-wiki-news-subwords-300 (999999 records): 1 million word vectors trained on Wikipe...
glove-twitter-100 (1193514 records): Pre-trained vectors based on  2B tweets,...
glove-twitter-200 (1193514 records): Pre-trained vectors based on 2B tweets, ...
glove-twitter-25 (1193514 records): Pre-trained vectors based on 2B tweets, ...
glove-twitter-50 (1193514 records): Pre-trained vectors based on 2B tweets, ...
glove-wiki-gigaword-100 (400000 records): Pre-trained vectors based on Wikipedia 2...
glove-wiki-gigaword-200 (400000 records): Pre-trained vectors based on Wikipedia 2...
glove-wiki-gigaword-300 (400000 records): Pre-trained vectors based on Wikipedia 2...
glove-wiki-gigaword-50 (400000 records): Pre-trained vectors based on Wikipedia 2...
word2vec-google-news-300 (3000000 records): Pre-trained vectors trained on a part of...
word2vec-ruscorpora-300 (184973 records): Word2vec Continuous Skipgram vectors tra...
```

Embedding – 실습

```
model = api.load("word2vec-google-news-300")
```

```
print(model.most_similar("cat"))
```

```
[('cats', 0.8099379539489746), ('dog', 0.760945737361908),
('kitten', 0.7464985251426697), ('feline', 0.7326234579086304),
```



성별 시제 국가-수도

```
print(model.most_similar_cosmul(positive=['Seoul', 'France'], negative=['Paris']))
```

[('South_Korea', 1.0531545877456665), ('Korea', 0.9701053500175476), ('South_Korean', 0.9288336634635925), ('Koreans', 0.9077238440513611), ('Japan', 0.9035927057266235), ('Korean', 0.903152346611023), ('Korea_ROK', 0.9019569754600525), ('SEOUL_NORTH', 0.8990684747695923), ('North_Korea', 0.894801914691925), ('SKorea', 0.8826186656951904)]

Embedding – 실습

```
print(model.most_similar_cosmul(positive=['father','woman'], negative=['man']))
print(model.most_similar_cosmul(positive=['brother','woman'], negative=['man']))
```

```
[('mother', 1.025557279586792), ('daughter', 0.991381824016571),]
[('sister', 0.9868288636207581), ('daughter', 0.9439969658851624)]
```

```
print(model.most_similar_cosmul(positive=['soju','mexico'], negative=['korea']))
print(model.most_similar_cosmul(positive=['soju','russia'], negative=['korea']))
```

[('tequila', 0.8992794156074524), ('mezcal', 0.8555493950843811), ..]
[('vodka', 0.8616750240325928), ('brandy', 0.8266340494155884), ('distilled_liquor', 0.8266003727912903),…]

Embedding – 데이터 살펴보기

- IMDB 데이터셋

감성 분류를 연습하기 위해 자주 사용하는 영어 데이터로 영화 사이트 IMDB의 리뷰 데이터가 있습니다.

이 데이터는 리뷰에 대한 텍스트와 해당 리뷰가 긍정인 경우 1을 부정인 경우 0으로 표시한 레이블로 구성된 데이터입니다.

스탠포드 대학교에서 2011년에 낸 논문에서 이 데이터를 소개하였으며, 당시 논문에서는 이 데이터를 훈련 데이터와 테스트 데이터를 50:50대 비율로 분할하여 88.89%의 정확도를 얻었다고 소개하고 있습니다.

Embedding – 데이터 살펴보기

- 데이터 다운로드

[함께 해봐요] **데이터셋 다운받기** use_embedding_layer.ipynb

```
01 from tensorflow.keras.datasets import imdb
02
03 num_words = 10000
04 (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
```

  - num_words 인자를 통해 사용할 단어의 개수를 조절(여기서는 10,000개만 사용)
  - 학습 데이터와 테스트 데이터는 5:5 비율로 나뉘어서 제공

- 데이터 확인
  - 데이터에서 확인할 수 있는 숫자는 빈번하게 사용되는 정도를 나타냄
  - 레이블 → **1(긍정), 0(부정)**

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 458, 4468, 66, 3941, 4, 173, 36,
256, 5, 25, 100, 43, 838, 112, 50, 670, 2, 9, 35, 480, 284, 5, 150, 4, 172,
... 생략 ...
-------
1
```

Embedding – 데이터 살펴보기

- 가장 빈번하게 사용되는 세 가지 단어 출력해보기
  - the, and, a
  - 포함? 제외? 고민해볼 수 있음
  - 포함시킨다면 어떤 방법으로 포함시킬 것인가 등

[함께 해봐요] IMDB 데이터셋에서 가장 빈번하게 사용되는 세 개의 단어

```
01  imdb_get_word_index = {}
02
03  for key, value in imdb.get_word_index().items():
04      imdb_get_word_index[value] = key
05
06  for i in range(1, 4):
07      print('{} 번째로 가장 많이 쓰인 단어 = {}'.format(i, imdb_get_word_index[i]))
```

```
1 번째로 가장 많이 쓰인 단어 = the
2 번째로 가장 많이 쓰인 단어 = and
3 번째로 가장 많이 쓰인 단어 = a
```

- **데이터의 길이가 전부 동일하도록 조정**해주기 위해 pad_sequences() 함수를 사용
  - 지정해준 길이보다 짧은 경우 0으로 채워넣음(zero padding), 긴 경우는 잘라냄

```
07  pad_X_train = pad_sequences(X_train, maxlen=max_len, padding = 'pre')
08  pad_X_test = pad_sequences(X_test, maxlen=max_len, padding = 'pre')
```

```
Before pad_sequences:  218
After pad_sequences:  500
pad_X_train:  [   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    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   14   22   16   43  530  973 1622 1385   65  458 4468
   66 3941    4  173   36  256    5   25  100   43  838  112   50  670
    2    9   35  480  284    5  150    4  172  112  167    2  336  385
   39    4  172 4536 1111   17  546   38   13  447    4  192   50   16
    6  147 2025   19   14   22    4 1920 4613  469    4   22   71   87
   12   16   43  530   38   76   15   13 1247    4   22   17  515   17
   12   16  626   18    2    5   62  386   12    8  316    8  106    5
    4 2223 5244   16  480   66 3785   33    4  130   12   16   38  619
    5   25  124   51   36  135   48   25 1415   33    6   22   12  215
   28   77   52    5   14  407   16   82    2    8    4  107  117 5952
   15  256    4    2    7 3766    5  723   36   71   43  530  476   26
  400  317   46    7    4    2 1029   13  104   88    4  381   15  297
   98   32 2071   56   26  141    6  194 7486   18    4  226   22   21
  134  476   26  480    5  144   30 5535   18   51   36   28  224   92
   25  104    4  226   65   16   38 1334   88   12   16  283    5   16
 4472  113  103   32   15   16 5345   19  178   32]
```

Embedding – 모델 구성하기

- Embedding층은 모델의 **첫 번째 층으로만 사용할 수 있으며, 주로 순환 신경망과 연결하여 사용**
  - **(batch_size, sequence_length)** 형태를 입력으로 받으며, **(batch_size, sequence_length, output_dim)** 형태를 출력

```
04  model = Sequential()
05  # 이 층은 모델의 제일 첫 번째 층으로만 사용할 수 있습니다.
06  # Flatten층을 사용하기 위해 input_length를 전달합니다.
07  model.add(Embedding(input_dim = num_words, output_dim = 32,
                        input_length = max_len))
```

  - input_dim(학습 데이터에서 사용한 단어의 개수), output_dim(임베딩 벡터 크기)
  - input_length 인자는 순환 신경망과 연결할 경우엔 사용하지 않음

Embedding – 모델 학습하고 평가하기

[함께 해봐요] **모델 학습시키기** use_embedding_layer.ipynb

```
01  history = model.fit(pad_X_train, y_train, batch_size = 32, epochs = 30,
                        validation_split = 0.2)
```

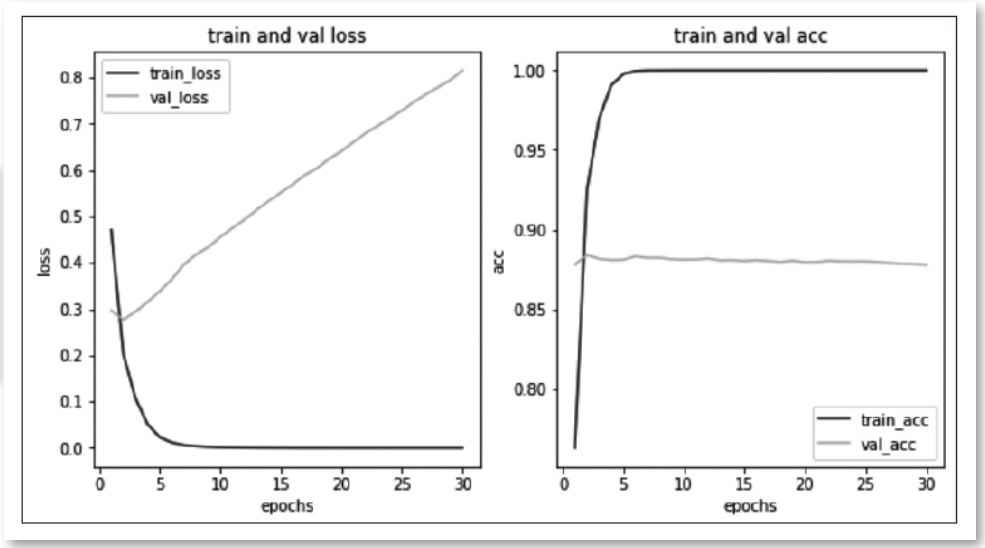
- validation_split 인자 사용
  - 학습 데이터의 끝에서 해당 비율만큼 떼어내어 검증 데이터셋으로 활용
  - 무작위로 20% 비율만큼 뽑아오는 것이 아닌 **단순하게 끝에서 떼어낸다**는 점을 주의

- 항상 결과를 확인하고, 학습 과정을 기록하는 것을 습관화할 것

[함께 해봐요] **모델 평가하기** use_embedding_layer.ipynb

```
01  model.evaluate(pad_X_test, y_test)
```

```
[0.8114458246806264, 0.87172]
```



Embedding – IMDB 전체소스 1

```
from keras.datasets import imdb

num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)


print(X_train.shape, y_train.shape) # (25000,) (25000,)
print(X_test.shape, y_test.shape) # (25000,) (25000,)

print(X_train[0])
print('-------')
print(y_train[0])



imdb_get_word_index = {}

for key, value in imdb.get_word_index().items():
    imdb_get_word_index[value] = key

for i in range(1, 50):
    print('{} 번째로 가장 많이 쓰인 단어 = {}'.format(i, imdb_get_word_index[i]))
```

Embedding – IMDB 전체소스 2

```python
from keras.preprocessing.sequence import pad_sequences

max_len = 500

print('Before pad_sequences: ', len(X_train[0]))

pad_X_train = pad_sequences(X_train, maxlen=max_len, padding = 'pre')
pad_X_test = pad_sequences(X_test, maxlen=max_len, padding = 'pre')

print('After pad_sequences: ', len(pad_X_train[0]))
print('pad_X_train: ', pad_X_train[0])
```

```
Before pad_sequences:  218
After pad_sequences:  500
pad_X_train:  [   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    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    0    0    0    0    0    0
    0    0    1   14   22   16   43  530  973 1622 1385   65  458 4468
   66 3941    4  173   36  256    5   25  100   43  838  112   50  670
    2    9   35  480  284    5  150    4  172  112  167    2  336  385
   39    4  172 4536 1111   17  546   38   13  447    4  192   50   16
    6  147 2025   19   14   22    4 1920 4613  469    4   22   71   87
   12   16   43  530   38   76   15   13 1247    4   22   17  515   17
   12   16  626   18    2    5   62  386   12    8  316    8  106    5
    4 2223 5244   16  480   66 3785   33    4  130   12   16   38  619
    5   25  124   51   36  135   48   25 1415   33    6   22   12  215
   28   77   52    5   14  407   16   82    2    8    4  107  117 5952
   15  256    4    2    7 3766    5  723   36   71   43  530  476   26
  400  317   46    7    4    2 1029   13  104   88    4  381   15  297
   98   32 2071   56   26  141    6  194 7486   18    4  226   22   21
  134  476   26  480    5  144   30 5535   18   51   36   28  224   92
   25  104    4  226   65   16   38 1334   88   12   16  283    5   16
 4472  113  103   32   15   16 5345   19  178   32]
```

Embedding – IMDB 전체소스 3

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

model = Sequential()

# 이 층은 모델의 제일 첫 번째 층으로만 사용할 수 있습니다.
Flatten 층을 사용하기 위해 input_length를 전달합니다.
model.add(Embedding(input_dim = num_words, output_dim = 32, input_length = max_len))
model.add(Flatten())
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

Embedding – IMDB 전체소스 4

```
model.summary()
```

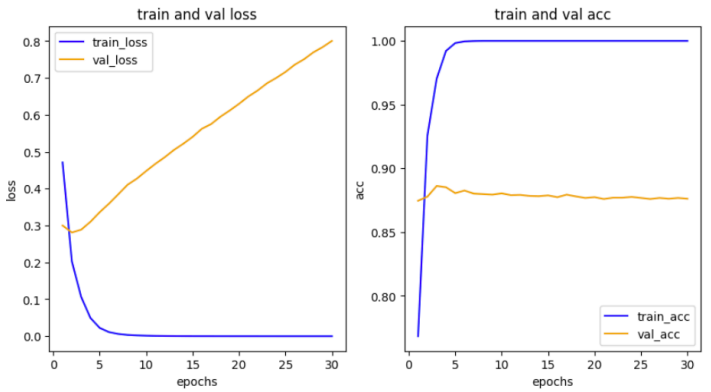
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 500, 32)           320000

 flatten (Flatten)           (None, 16000)             0

 dense (Dense)               (None, 1)                 16001

=================================================================
Total params: 336001 (1.28 MB)
Trainable params: 336001 (1.28 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
history = model.fit(pad_X_train, y_train,
                    batch_size = 32,
                    epochs = 30,
                    validation_split = 0.2)
```

```
model.evaluate(pad_X_test, y_test) # [0.8045057058334351, 0.8700399994850159]
```

Embedding – IMDB 전체소스 5

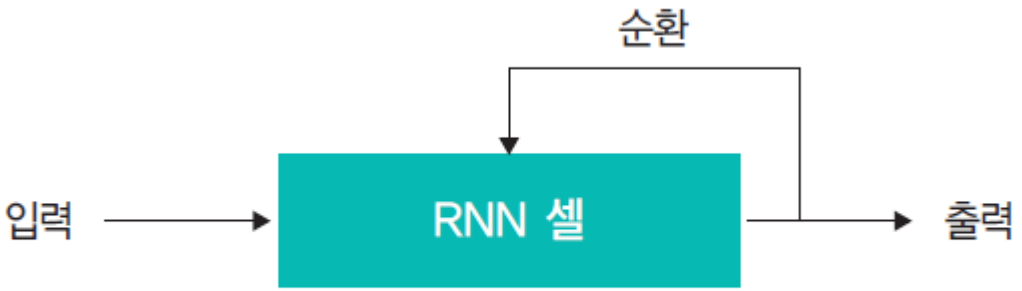
```
import matplotlib.pyplot as plt

his_dict = history.history
loss = his_dict['loss']
val_loss = his_dict['val_loss']

epochs = range(1, len(loss) + 1)
fig = plt.figure(figsize = (10, 5))

# 훈련 및 검증 손실 그리기
ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

acc = his_dict['acc']
val_acc = his_dict['val_acc']

# 훈련 및 검증 정확도 그리기
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
ax2.set_title('train and val acc')
ax2.set_xlabel('epochs')
ax2.set_ylabel('acc')
ax2.legend()

plt.show()
```



RNN – 원리 이해하기

- Embedding층은 데이터의 표현을 학습하여 데이터 사전을 구축하는 것으로 쉽게 이해할 수 있음
  - 유사 단어를 비슷한 공간에 매핑하지만, **시퀀스 데이터의 중요한 특성인 순서와 맥락을 고려하지 않음**
  - “Dense vs Conv” 차이를 떠올려보자

- 순환 신경망은 완전연결층, 컨볼루션 신경망의 반대되는 개념으로 설명할 수 있음
  - 완전연결층과 컨볼루션 신경망은 **피드 포워드 네트워크(feed-forward network)**
  - 피드 포워드 네트워크는 출력값이 오직 마지막 층인 출력층을 향함
  - 하지만 순환 신경망은 출력값이 출력층을 향하면서도 동시에 **현재층의 다음 값으로 사용**

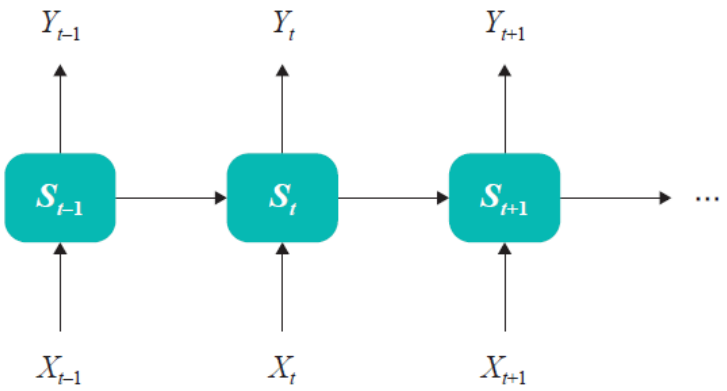


[그림 6-3] 순환 신경망의 구조-1

RNN – 원리 이해하기

- RNN 셀
  - 순환 신경망의 노드가 출력값을 반환하는 동시에 이전 상태(state)를 기억하는 메모리 역할을 수행
  - **은닉 상태(hidden state)**

- 다음 그림에서 x는 입력, y는 출력, t는 현재 시점을 의미
- 의사코드에서 output_t가 state_t의 값을 변환시키는 것을 확인
  - state_t는 activation_func(활성화 함수)에서 사용되고 있음

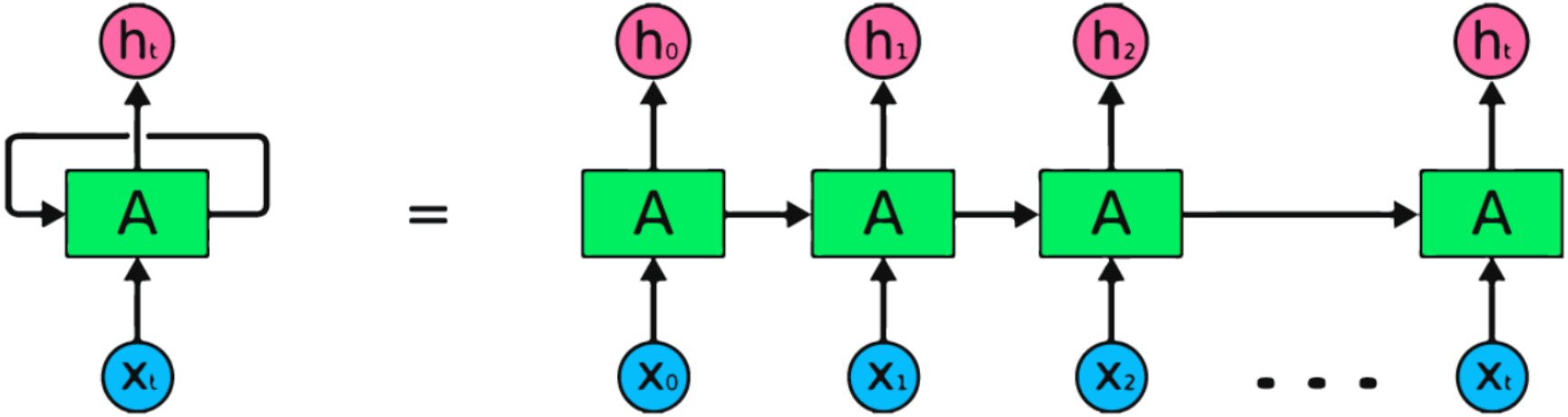


[그림 6-4] 순환 신경망의 구조-2

예시: 순환 신경망을 표현한 의사코드

```
state_t = 0  # 초기 상태

# 각 시점에 해당하는 입력을 반복합니다.
for input_t in input_sequence:
    # 입력과 은닉상태를 활성화 함수에 통과시킵니다.
    output_t = activation_func(input_t, state_t)
    # 출력값은 다음 시점을 위한 은닉 상태가 됩니다.
    state_t = output_t
```

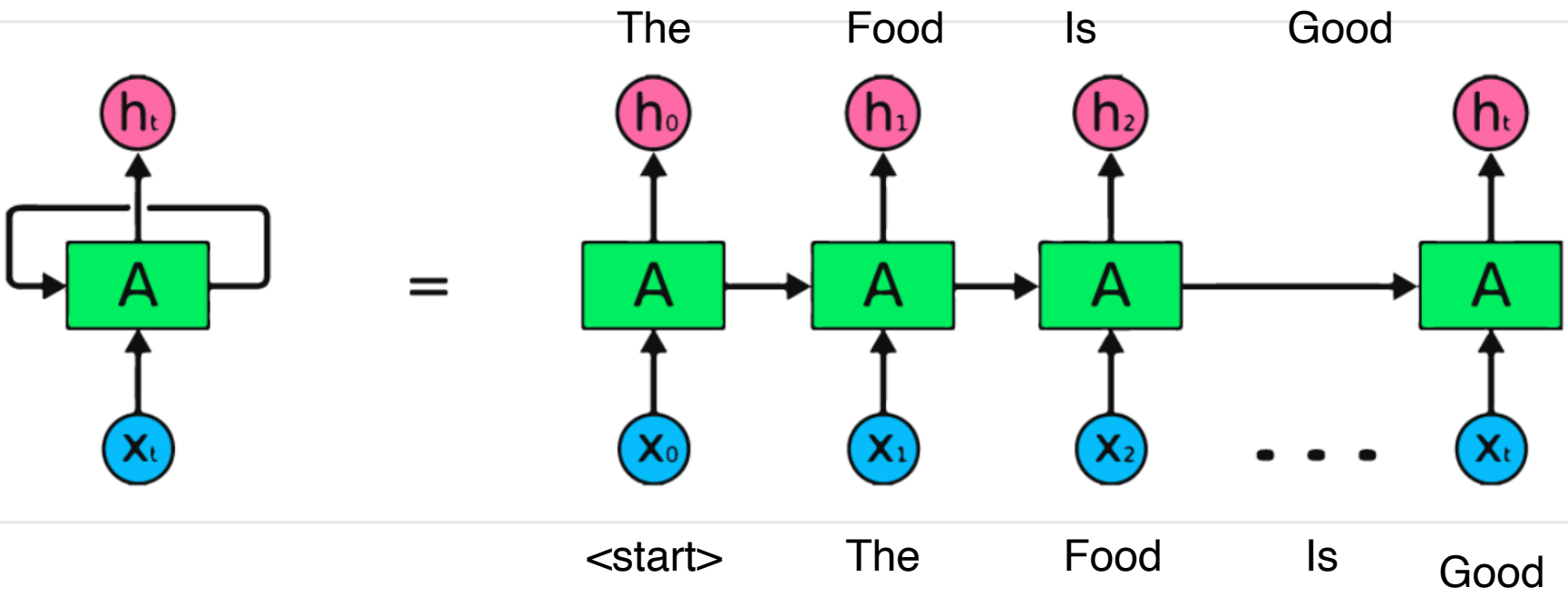
RNN – 원리 이해하기



RNN – 원리 이해하기

은닉상태(hidden state)

Positive



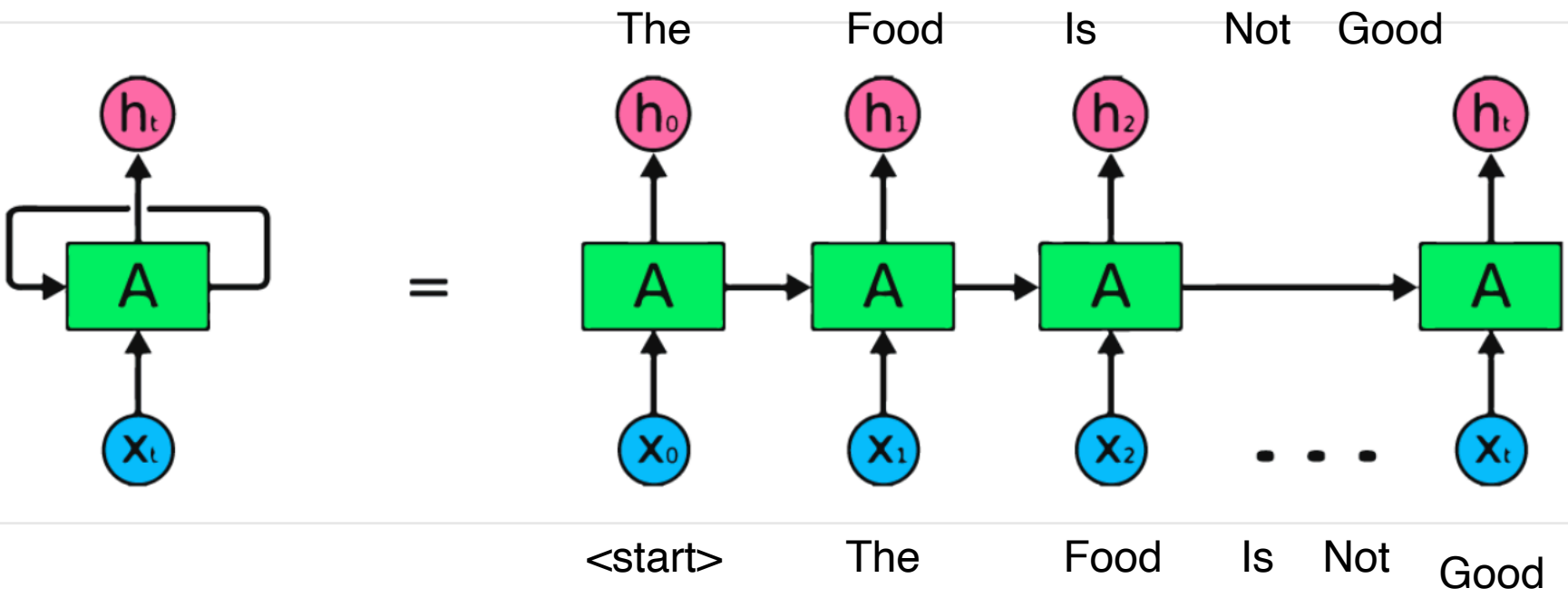
Input The food is good

Output : Positive

RNN – 원리 이해하기

은닉상태(hidden state)

Negative



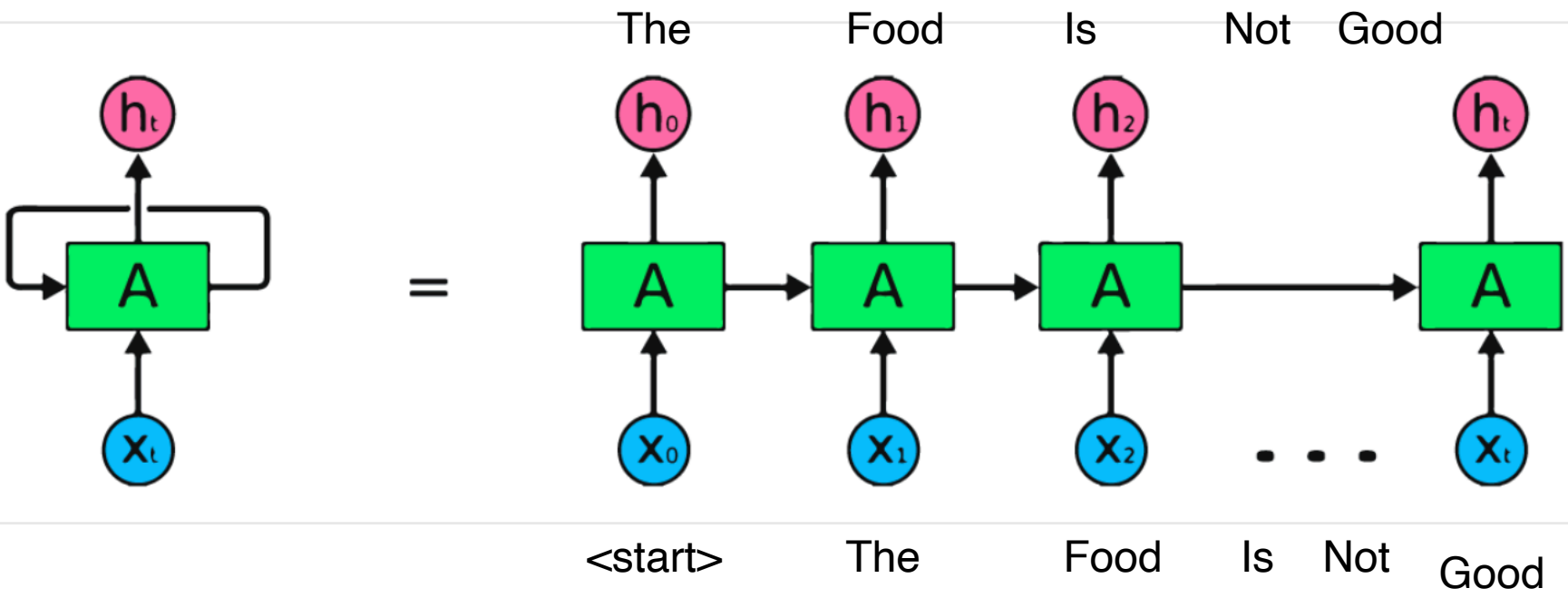
Input The food is not good

Output : Negative

RNN – 원리 이해하기

은닉상태(hidden state)

Negative



Input The food is not good

Output : Negative

RNN – IMDB 데이터셋으로 RNN 학습시키기 1

```
from keras.datasets import imdb

# 1 ~ 10,000 빈도 순위에 해당하는 단어만 사용합니다.
num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words = num_words)


from keras.preprocessing.sequence import pad_sequences

# 각 데이터의 최대 길이를 500으로 동일하게 맞춰줍니다.
max_len = 500

pad_X_train = pad_sequences(X_train, maxlen=max_len)
pad_X_test = pad_sequences(X_test, maxlen=max_len)
```

RNN – IMDB 데이터셋으로 RNN 학습시키기 2

```
from keras.models import Sequential
from keras.layers import SimpleRNN, Dense, Embedding

model = Sequential()

model.add(Embedding(input_dim = num_words, output_dim = 32))

# 새로운 인자 3개가 사용되었습니다.
# return_sequences, dropout, recurrent_dropout
model.add(SimpleRNN(32, return_sequences = True, dropout = 0.15,
       recurrent_dropout = 0.15))
model.add(SimpleRNN(32))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics = ['acc'])

# model.summary()
history = model.fit(pad_X_train, y_train,
                    batch_size = 32, epochs = 15,
                    validation_split = 0.2)
```

RNN – IMDB 데이터셋으로 RNN 학습시키기 3

```
import matplotlib.pyplot as plt

his_dict = history.history
loss = his_dict['loss']
val_loss = his_dict['val_loss']

epochs = range(1, len(loss) + 1)
fig = plt.figure(figsize = (10, 5))

# 훈련 및 검증 손실 그리기
ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

acc = his_dict['acc']
val_acc = his_dict['val_acc']

# 훈련 및 검증 정확도 그리기
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
ax2.set_title('train and val acc')
ax2.set_xlabel('epochs')
ax2.set_ylabel('acc')
ax2.legend()

plt.show()
```

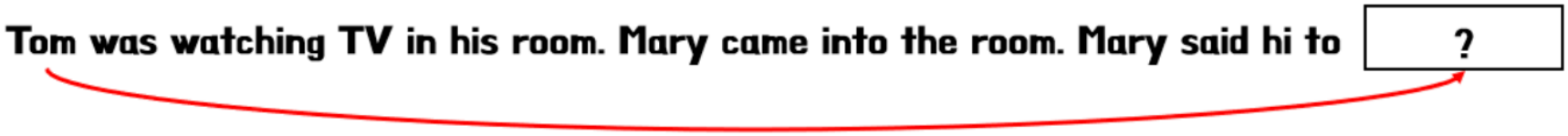
RNN – 여러 개 쌓아보기(실습)

- SimpleRNN층을 여러 개 연결하기 위해서는 추가 인자를 설정
  - **recurrent_sequence**
  - True일 경우, RNN 셀의 전체 상태를 반환
  - 드롭아웃 사용을 위해 **dropout, recurrent_dropout** 인자를 사용
  - 제공되는 코드를 통해 IMDB 데이터셋에 적용해보세요

```
16  model = Sequential()
17  model.add(Embedding(input_dim = num_words, output_dim = 32))
18  # 새로운 인자 세 개가 사용되었습니다.
19  model.add(SimpleRNN(32, return_sequences = True, dropout = 0.15,
                        recurrent_dropout = 0.15))
20  model.add(SimpleRNN(32))
```

RNN 문제점

기울기 소실 문제



장기 기억 필요한 문제의 예시

장기기억 소실 문제:

장기 의존성을 학습하기 어렵습니다.
시퀀스가 길어질수록 초기 입력 정보의 영향력이 점점 줄어들어,
학습이 효과적으로 이루어지지 않는 경우가 많습니다.

개선된 RNN 모델
이러한 문제들을 해결하기 위해 LSTM(Long Short-Term Memory)과 GRU(Gated Recurrent Units)과 같은 구조가 개발되었습니다.
이 구조들은 장기 의존성 문제를 더 잘 처리하도록 설계되어, 복잡한 시퀀스 데이터를 효과적으로 학습할 수 있습니다.

LSTM – 원리 이해하기

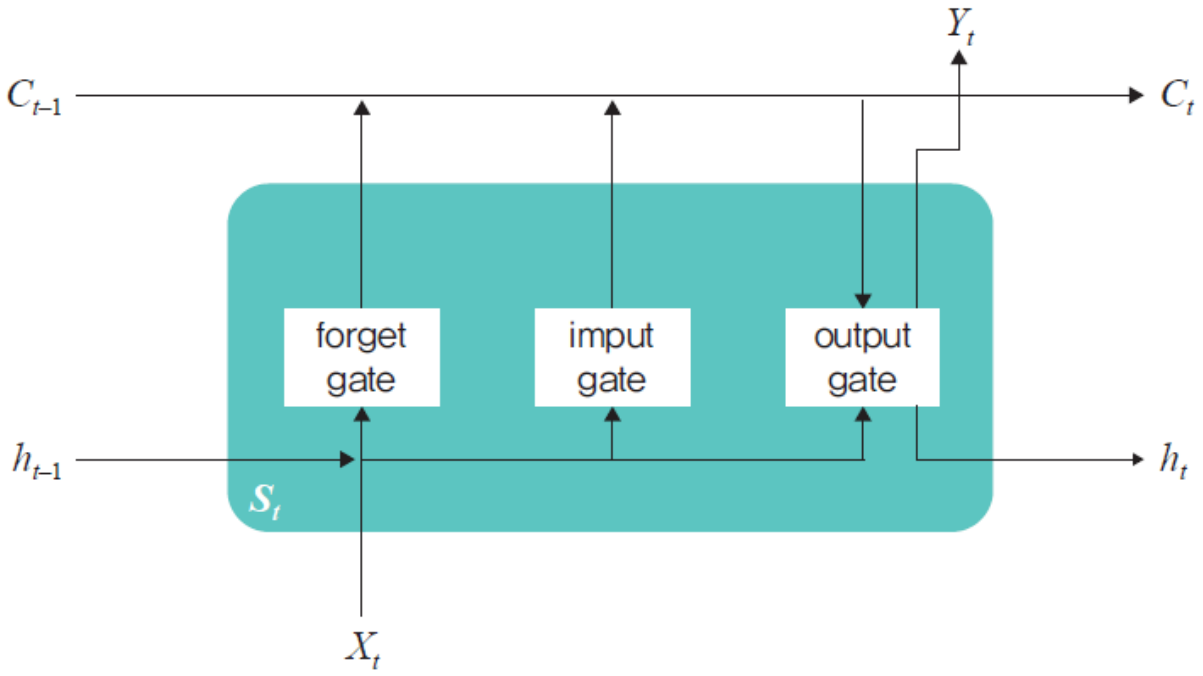
- 단순한 순환 신경망의 문제점
  - ‘시점이 흐를수록 지속해서 기억하지 못한다’
  - **그래디언트 손실 문제**
  - **이러한 문제는 chatGPT와 같은 LLM(Large Language Model)에서도 아직 근본적으로 해결되지 않았음**

- 이를 해결하기 위해 1997년 고안된 방법
  - **LSTM(Long Short-Term Memory)**
  - Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

- 여전히 많은 사례에서 사용되고 있으며, 이를 변형한 여러 가지 방법이 존재

LSTM – 원리 이해하기

- LSTM의 핵심은 **정보를 여러 시점에 걸쳐 나르는 장치**가 추가되었다는 것
  - 그래디언트를 보존할 수 있어 그래디언트 손실 문제가 발생하지 않도록 도와줌
  - 아래 그림에서 C로 표현된 **'Cell State'**, h는 hidden state
  - 정보 나르는 것을 도와줄 세 개의 게이트: **forget gate, input gate, output gate**



[그림 6-6] LSTM의 구조

LSTM – 원리 이해하기

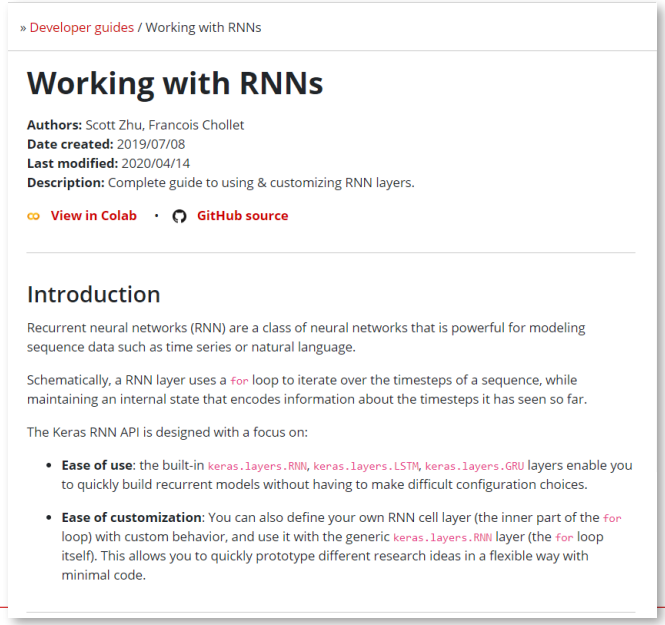
게이트	내용[12]	
forget_gate	수식	f_t = sigmoid(dot(x_t, Wf) + dot(h_{t-1}, Uf) + bf)[13]
	설명	'Cell state'가 나르는 정보 중, 관련 없는 정보를 시그모드 함수를 사용하여 제거합니다(f_t). 'Cell state'는 여기서 생성된 값과 원소별 곱을 통해 정보를 제거하게 됩니다. 곱은 아래 'input_gate'의 수식에서 볼 수 있습니다(f_t * C_{t-1}).
input_gate	수식	i_t = sigmoid(dot(x_t, Wi) + dot(h_{t-1}, Ui) + bi) C_t = f_t * C_{t-1} + i_t * tanh(dot(x_t, Wc) + dot(h_{t-1}, Uc) + bc)
	설명	input_gate는 두 가지 작업을 수행합니다. 첫 번째, 현재 시점의 정보(x_t)와 이전 시점의 상태(h_{t-1})에 시그모이드 함수를 활용하여 어떤 정보를 업데이트할지 결정합니다. 두 번째, 현재 시점의 정보와 이전 시점의 상태에 tanh 함수를 활용하여 새로운 정보를 만듭니다. 이 둘을 곱한 뒤, 'forget gate'를 통해 걸러진 정보와 더해져 현재 시점의 'Cell state'를 만들게 됩니다(C_t).
output_gate	수식	o_t = sigmoid(dot(x_t, Wo) + dot(h_{t-1}, Uo) + bo) h_t = o_t * tanh(C_t)
	설명	'output gate'는 출력값과 현재 시점의 상태 h_t를 만듭니다. h_t는 현재 시점의 정보와 이전 시점의 상태에 시그모이드 함수를 통과시켜 얻은 값과 tanh 함수를 통과한 'Cell state' 값을 곱해 만들어집니다. 또, h_t는 그림의 h_t와 동일하며, 결괏값을 만들기 위해 활성화 함수를 통과한 h_t는 그림의 y_t와 동일합니다.

LSTM – 원리 이해하기

- 신경망 공부를 시작했지만, 수학은 너무 어려워…
  - (우리에게 중요) 식을 전부 기억할 필요는 없음
  - 다만, 모든 연산이 ‘Cell State’를 중심으로 이루어진다는 것과 LSTM의 핵심적인 기능을 기억!

- LSTM은 워낙 유명한 모델이라 이를 잘 설명한 많은 글들이 있습니다.
  - 다른 표현들로 설명되어 있으니, 종합하여 나만의 것으로 만드세요.
  - 이해하고 사용하는 것과 이해하지 않고 사용하는 것은 엄청난 차이를 가져다 줍니다.

- 데이터를 다뤄봅시다!
  - 추가로, 케라스 공식 홈페이지에도 좋은 튜토리얼이 존재합니다
    - https://keras.io/guides/working_with_rnns/


» Developer guides / Working with RNNs

Working with RNNs

Authors: Scott Zhu, Francois Chollet
Date created: 2019/07/08
Last modified: 2020/04/14
Description: Complete guide to using & customizing RNN layers.

View in Colab • GitHub source

Introduction

Recurrent neural networks (RNN) are a class of neural networks that is powerful for modeling sequence data such as time series or natural language.

Schematically, a RNN layer uses a `for` loop to iterate over the timesteps of a sequence, while maintaining an internal state that encodes information about the timesteps it has seen so far.

The Keras RNN API is designed with a focus on:

- **Ease of use**: the built-in `keras.layers.RNN`, `keras.layers.LSTM`, `keras.layers.GRU` layers enable you to quickly build recurrent models without having to make difficult configuration choices.
- **Ease of customization**: You can also define your own RNN cell layer (the inner part of the `for` loop) with custom behavior, and use it with the generic `keras.layers.RNN` layer (the `for` loop itself). This allows you to quickly prototype different research ideas in a flexible way with minimal code.

LSTM – IMDB 데이터셋으로 LSTM 학습시키기 1

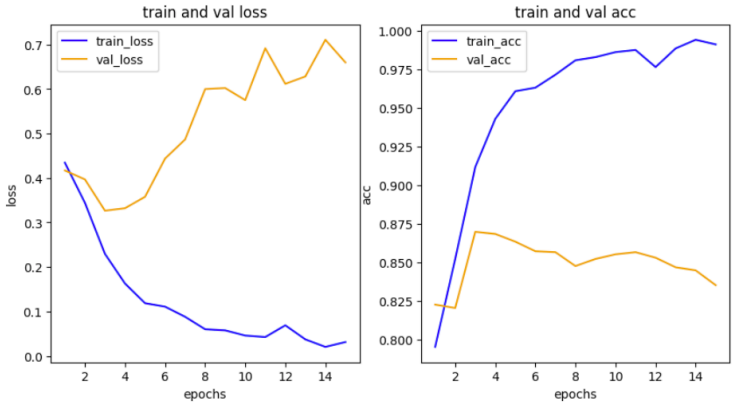
```
from tensorflow.keras.datasets import imdb

num_words = 10000
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

from tensorflow.keras.preprocessing.sequence import pad_sequences

max_len = 350

pad_X_train = pad_sequences(X_train, maxlen=max_len)
pad_X_test = pad_sequences(X_test, maxlen=max_len)

print(len(pad_X_train[0]))

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Embedding

model = Sequential()

model.add(Embedding(input_dim = num_words, output_dim = 32))
model.add(LSTM(64, return_sequences = True))
model.add(LSTM(32))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics = ['acc'])
```

LSTM – IMDB 데이터셋으로 LSTM 학습시키기 2

```
model.summary()

history = model.fit(pad_X_train, y_train,
                    batch_size = 32, epochs = 15,
                    validation_split = 0.2)
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, None, 32)          320000

 lstm_3 (LSTM)               (None, None, 64)          24832

 lstm_4 (LSTM)               (None, 32)                12416

 dense_3 (Dense)             (None, 1)                 33

=================================================================
Total params: 357281 (1.36 MB)
Trainable params: 357281 (1.36 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
import matplotlib.pyplot as plt

his_dict = history.history
loss = his_dict['loss']
val_loss = his_dict['val_loss']

epochs = range(1, len(loss) + 1)
fig = plt.figure(figsize = (10, 5))

# 훈련 및 검증 손실 그리기
ax1 = fig.add_subplot(1, 2, 1)
ax1.plot(epochs, loss, color = 'blue', label = 'train_loss')
ax1.plot(epochs, val_loss, color = 'orange', label = 'val_loss')
ax1.set_title('train and val loss')
ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend()

acc = his_dict['acc']
val_acc = his_dict['val_acc']

# 훈련 및 검증 정확도 그리기
ax2 = fig.add_subplot(1, 2, 2)
ax2.plot(epochs, acc, color = 'blue', label = 'train_acc')
ax2.plot(epochs, val_acc, color = 'orange', label = 'val_acc')
ax2.set_title('train and val acc')
ax2.set_xlabel('epochs')
ax2.set_ylabel('acc')
ax2.legend()

plt.show()
```



LSTM – 네이버쇼핑 댓글 악플체크 모델-1

```
import urllib.request
urllib.request.urlretrieve('https://raw.githubusercontent.com/bab2min/corpus/master/sentiment/
naver_shopping.txt', 'shopping.txt')


import pandas as pd
import numpy as np

# pandas로 데이터파일 읽기 + 컬럼에 제목달기
raw = pd.read_table('shopping.txt', names=['rating', 'review'])
print(raw)
```

```
        rating                                                 review
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
...        ...                                                ...
199995       2                                장마라그런가!!! 달지않아요
199996       5  다이슨 케이스 구매했어요 다이슨 슈퍼소닉 드라이기 케이스 구매했어요가격 괜찮고 배송...
199997       5                    로드샵에서 사는것보다 세배 저렴하네요 ㅜㅜ 자주이용할께요
199998       5                                  넘이쁘고 쎄련되보이네요~
199999       5   아직 사용해보지도않았고 다른 제품을 써본적이없어서 잘 모르겠지만 ㅎㅎ 배송은 빨랐습니다

[200000 rows x 2 columns]
```

LSTM – 네이버쇼핑 댓글 악풀체크 모델-2

```python
# label열을 추가
raw['label'] = np.where(raw['rating'] > 3, 1, 0) # rating이 3보다 크면 1, 그렇지 않으면 0(악플)
print(raw)
```

```
        rating                                             review  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
...        ...                                                ...    ...
199995       2                                   장마라그런가!!! 달지않아요      0
199996       5  다이슨 케이스 구매했어요 다이슨 슈퍼소닉 드라이기 케이스 구매했어요가격 괜찮고 배송...      1
199997       5                   로드샵에서 사는것보다 세배 저렴하네요 ㅜㅜ 자주이용할께요      1
199998       5                                     넘이쁘고 쎄련되보이네요~      1
199999       5   아직 사용해보지도않았고 다른 제품을 써본적이없어서 잘 모르겠지만 ㅎㅎ 배송은 빨랐습니다      1

[200000 rows x 3 columns]
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-3

```
# 한글전처리 1. 특수문자 제거
raw['review'] = raw['review'].str.replace('[^ㄱ-ㅎ ㅏ-ㅣ가-힣0-9 ]', '', regex=True)
 #한글,숫자,스페이스 외는 제거


# 한글전처리 2. 중복값 제거
raw.drop_duplicates( subset='review', inplace=True )
print(raw)
```

```
        rating                                                 review  label
0            5                                              배공빠르고 굿       1
1            2                                택배가 엉망이네용 저희집 밑에층에 말도없이 놔두고가고       0
2            5  아주좋아요 바지 정말 좋아서개 더 구매했어요 이가격에 대박입니다 바느질이 조금 엉성...       1
3            2  선물용으로 빨리 받아서 전달했어야 하는 상품이었는데 머그컵만 와서 당황했습니다 전화...       0
4            5                          민트색상 예뻐요 옆 손잡이는 거는 용도로도 사용되네요 ㅎㅎ       1
...        ...                                                ...     ...
199995       2                                        장마라그런가 달지않아요       0
199996       5  다이슨 케이스 구매했어요 다이슨 슈퍼소닉 드라이기 케이스 구매했어요가격 괜찮고 배송...       1
199997       5                            로드샵에서 사는것보다 세배 저렴하네요 ㅜㅜ 자주이용할께요       1
199998       5                                        넘이쁘고 쎄련되보이네요       1
199999       5   아직 사용해보지도않았고 다른 제품을 써본적이없어서 잘 모르겠지만 ㅎㅎ 배송은 빨랐습니다       1

[199391 rows x 3 columns]
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-4

```
# bag of words
uniqueWord = raw['review'].tolist()
uniqueWord = ''.join(uniqueWord)
print(uniqueWord[0:100])

uniqueWord = list(set(uniqueWord))
uniqueWord.sort()
print(uniqueWord[0:100])
```

```
배공빠르고 굿택배가 엉망이네용 저희집 밑에층에 말도없이 놔두고가고아주좋아요 바지 정말 좋아서개 더 구매했어요 이가격에 대박입니다 바느질이 조금
[' ', 'ㄱ', 'ㄲ', 'ㄳ', 'ㄴ', 'ㄵ', 'ㄶ', 'ㄷ', 'ㄸ', 'ㄹ', 'ㄺ', 'ㄻ', 'ㄼ', 'ㄽ', 'ㄾ', 'ㄿ', 'ㅀ', 'ㅁ', 'ㅂ', 'ㅃ'
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-5

```
# 문자단위로 정수로 변환
from keras.preprocessing.text import Tokenizer

tokenizer = Tokenizer(char_level=True, oov_token='<OOV>')  # 신조어는 <OOV>로 표시

wordlist = raw['review'].tolist()
tokenizer.fit_on_texts(wordlist)

print(tokenizer.word_index) # 문자를 정수로 변환함
```

```
{'<OOV>': 1, ' ': 2, '요': 3, '이': 4, '고': 5, '다': 6, '아': 7, '는': 8, '어': 9, '하': 10, '니': 11, '가': 12,
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-6

```
# 데이터셋을 정수로 변환하기
train_seq = tokenizer.texts_to_sequences(wordlist)
print(train_seq[0:100]) #일부만 출력
```

```
[[40, 318, 89, 88, 5, 2, 292], [286, 40, 12, 2, 394, 216, 4, 16, 47, 2, 84, 551, 215, 2, 504, 17, 870, 17,
```

```
#정답(label)도 리스트로 변경
Y = raw['label'].tolist()
print(Y[0:10])
```

```
#새로운열에 길이 정보 추가
raw['length'] = raw['review'].str.len()
```

```
#padding 처리
#제일 긴 문장은 몇자일까?
print(raw.head())
print(raw.describe())
```

```
   rating                                                 review  label  length
0       5                                              배공빠르고 굿      1       7
1       2                                택배가 엉망이네용 저희집 밑에층에 말도없이 놔두고가고      0      29
2       5  아주좋아요 바지 정말 좋아서개 더 구매했어요 이가격에 대박입니다 바느질이 조금 엉성...      1
3       2  선물용으로 빨리 받아서 전달했어야 하는 상품이었는데 머그컵만 와서 당황했습니다 전화...      0
4       5                          민트색상 예뻐요 옆 손잡이는 거는 용도로도 사용되네요 ㅎㅎ      1      3

              rating          label         length
count  199391.000000  199391.000000  199391.000000
mean        3.227056       0.500399      37.829275
std         1.645602       0.500001      27.816591
min         1.000000       0.000000       1.000000
25%         2.000000       0.000000      17.000000
50%         4.000000       1.000000      28.000000
75%         5.000000       1.000000      53.000000
max         5.000000       1.000000     140.000000
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-7

```
from keras.preprocessing.sequence import pad_sequences

X = pad_sequences(train_seq, maxlen=100) #길이가 100보다 길면 뒷부분은 cut

from sklearn.model_selection import train_test_split
trainX, valX, trainY, valY = train_test_split(X, Y, test_size=0.2, random_state=42)

print(len(trainX)) # 159,512
print(len(valX)) #39,879
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-8

```
# trainY와 valY를 numpy 배열로 변환
trainY = np.array(trainY)
valY = np.array(valY)

import tensorflow as tf
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

model = Sequential()

model.add(Embedding(len(tokenizer.word_index)+1, 16))
model.add(LSTM(64, return_sequences = True))
model.add(LSTM(32))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer='adam',
              loss = 'binary_crossentropy',
              metrics = ['acc'])


model.summary()

history = model.fit(trainX,trainY,
                    batch_size = 32, epochs = 2,
                    validation_data=(valX, valY))
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-9

```
model.save('naver_review.h5') # 모델 저장
```

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

# Load the model
new_model = load_model('naver_review.h5')
```

LSTM – 네이버쇼핑 댓글 악플체크 모델-10

```
def sentiment_predict(new_reviews):

  new_sequences = tokenizer.texts_to_sequences(new_reviews)
  new_padded = pad_sequences(new_sequences, maxlen=100)

  predictions = new_model.predict(new_padded)

  predicted_classes = ['긍정' if prob > 0.5 else '부정' for prob in predictions.flatten()]

  # Print predicted classes
  return predicted_classes



print(sentiment_predict(["여기 상품 정말 좋네요! 추천합니다."]))     # 긍정
print(sentiment_predict(["판매자님... 너무 짱이에요.. 대박나삼"]))  # 긍정
print(sentiment_predict(["ㅁㄴㅇㄹㄴㅇㄹㄴㅇ리뷰쓰기도 귀찮아"]))      # 부정
print(sentiment_predict(["별로에요. 기대했던 것보다 못해요."]))      # 부정
print(sentiment_predict(["진짜 배송도 늦고 개짜증나네요. 뭐 이런 걸 상품이라고 만듬?"])) #부정
```

RECAP

1. 순환 신경망은 **시퀀스 또는 시계열 데이터 처리에 특화**되어 있습니다.
2. Embedding층은 **수많은 단어(또는 데이터)를 표현**할 수 있습니다. 항상 모델의 첫 번째 층으로만 사용할 수 있습니다.
3. Embedding층은 **단어의 관계와 맥락을 파악할 수 없습니다**. 이를 해결하기 위해 사용되는 것이 SimpleRNN층입니다. SimpleRNN층은 순환 신경망의 가장 기본적인 형태를 나타내며, 출력값의 업데이트를 위해 **이전 상태를 사용**합니다.
4. SimpleRNN층은 **그래디언트 손실 문제를 야기**합니다. 이를 해결하기 위해 고안된 것이 LSTM입니다. LSTM은 **과거의 정보를 나르는 'Cell State'**를 가지고 있으며, 정보를 제거 또는 제공하기 위한 input_gate, forget_gate, output_gate를 보유하고 있습니다.