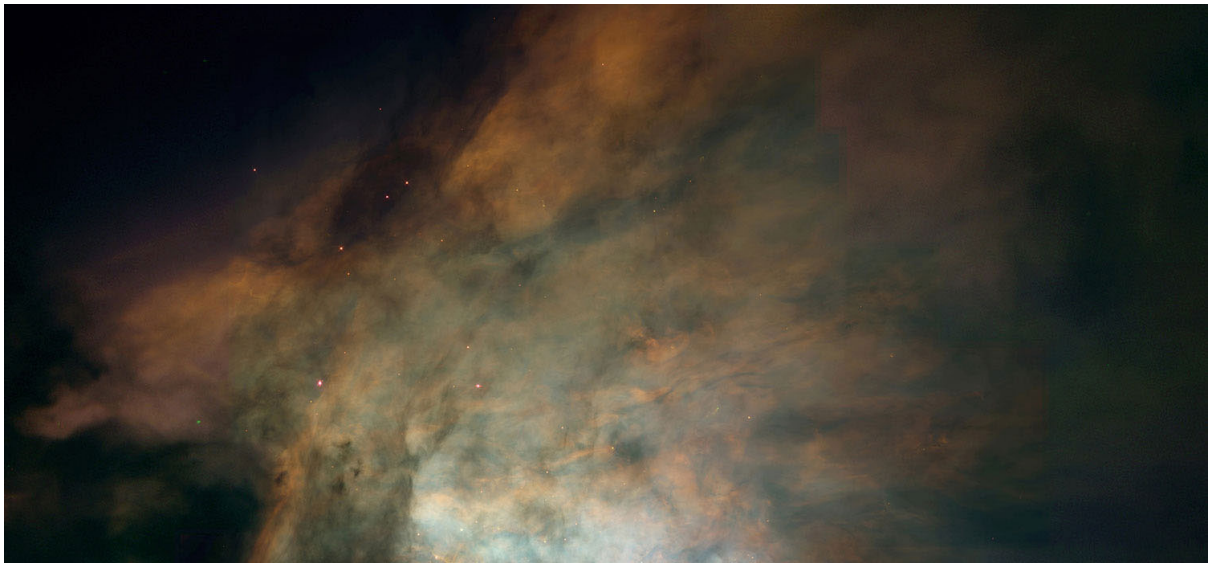


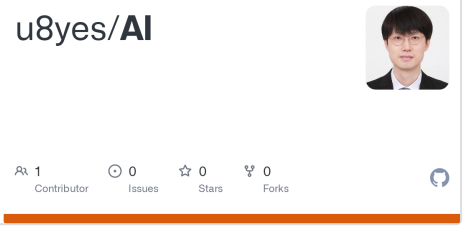
Day62; 20221202

날짜	@2022년 12월 2일
유형	@2022년 12월 2일
태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

https://github.com/u8yes/ai



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9ffedcec-1fc9-4999-b32e-12a6848a2fbe/data-02-stock_daily.csv

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9a54b225-bcbb-4831-b063-b2c913b34967/02_IMDB_LSTM_20221202.ipynb

tweed

미국·영국[twi:d] 영국식

1 트위드(간간이 다른 색깔의 올이 섞여 있는 두꺼운 모직 천)

2 트위드(로 만든) 옷

영어사전 다른 뜻 1

MIXXO

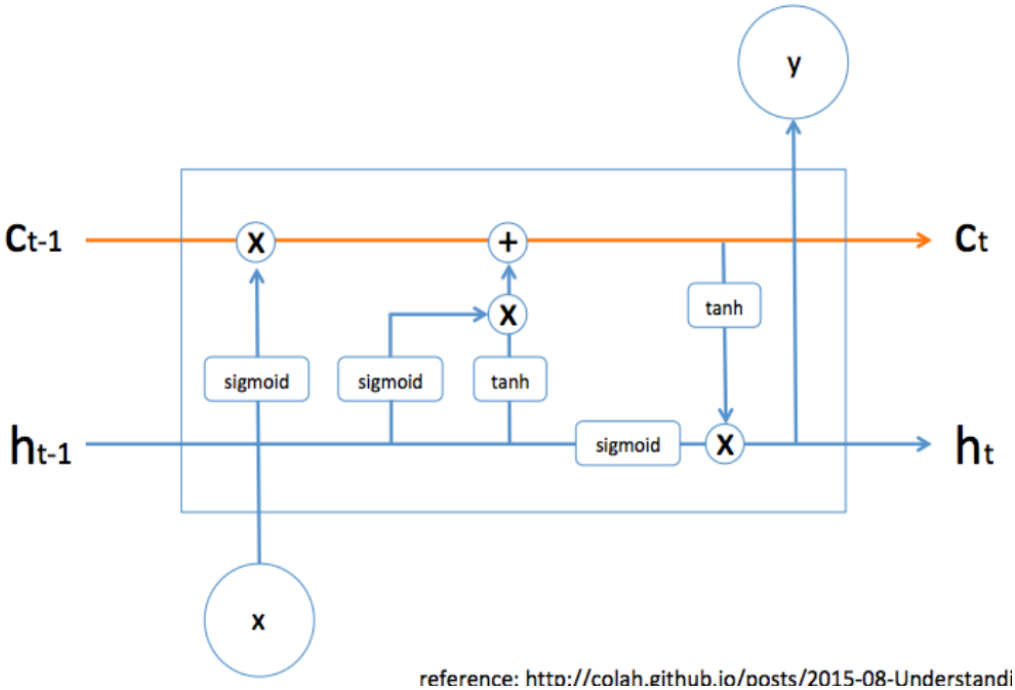
ID SALE 아우터 상의 하의 원피스 II

TWEED

자켓 상의 하의

순환신경망(Recurrent Neural Network)

LSTM(Long Short-Term Memory)



reference: http://colah.github.io/posts/2015-08-Understandi

1단어 처리되는 주기를 순환신경망에서는 '타입스텝'이라고 부른다.

셀은 메모리다.

C - cell

적절하게 삭제할 것 삭제하고, 적정수준을 기억해줌(예: 50%정도만 기억)

x 모양은 삭제시키는 logic, +모양은 과거의 기억을 보완을 시켜서 섞음

이전의 기억이 희미해지는 것을 극복시킴.

삭제할 때도 sigmoid 를 통해 1 미만으로 증폭을 막아줌.

남는 기억도 sigmoid로 1 미만으로 만들어주고 유지시켜줌. 다음 기억에 tahn을 거쳐서 버무려줌, 그리고 출력.

LSTM(Long Short-Term Memory)은 이렇게 이전 기억을 유지시킴.

SimpleRNN을 LSTM으로만 바꿔주면 됨.

(아래는 어제 자료)

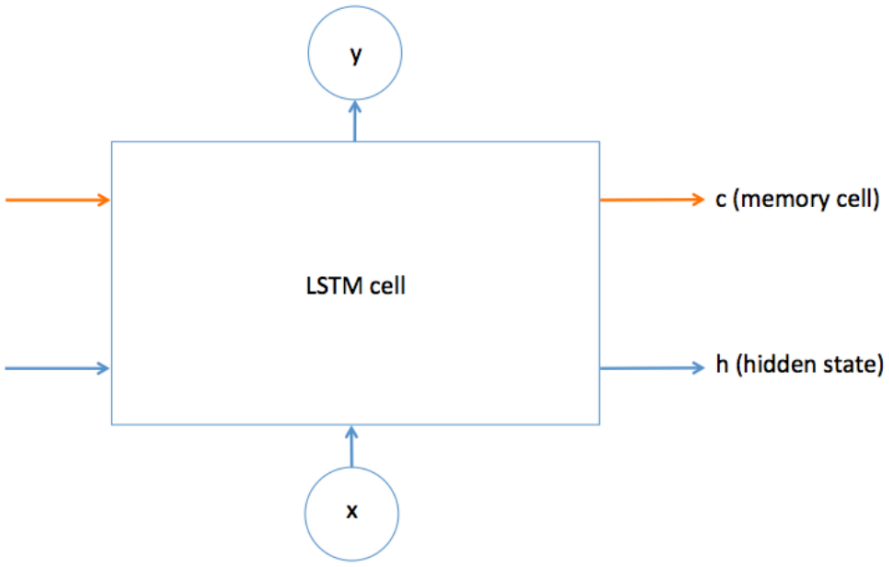
단어 임베딩 사용하기

```
model2 = keras.Sequential()

model2.add(keras.layers.Embedding(500, 16, input_length=100))
# 500개의 one-hot을 16개로 줄여줌.
model2.add(keras.layers.SimpleRNN(8)) # SimpleRNN을 LSTM으로만 바꿔주면 앞으로의 진도는 거의 끝
model2.add(keras.layers.Dense(1, activation='sigmoid'))
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           8000

 simple_rnn_1 (SimpleRNN)    (None, 8)                 200

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 8,209
Trainable params: 8,209
Non-trainable params: 0
_________________________________________________________________
```

LSTM(Long Short-Term Memory)

- 텐서플로우 사용 시, 이미 LSTM은 구현되어 있기 때문에 직접 위 그림을 구현할 필요는 없음.
- 텐서플로우 LSTM을 사용 시 아래 그림만 잘 이해하셔도, 사용에 큰 무리가 없음.



hidden state는 희미해짐

memory cell 부분은 기억하고 삭제만 해주는 부분.

임베딩(오늘 자료)

순환 신경망 만들기_20221202

]:
```
from tensorflow import keras

model = keras.Sequential()

model.add(keras.layers.Embedding(500, 16, input_length=100))
model.add(keras.layers.LSTM(8))
# 신경망 - Dense, CNN - Conv2D, RNN - RimpleRNN()
# default가 tahn으로 -1~1로 해놔서 달아놓을 필요 없다.
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           8000

 lstm (LSTM)                 (None, 8)                 800

 dense (Dense)               (None, 1)                 9

=================================================================
Total params: 8,809
Trainable params: 8,809
Non-trainable params: 0
_________________________________________________________________
```

아래의 코드는 실제 동작되는 코드가 아니라 의사 코드(pseudo-code)로 임베딩의 개념 이해를 돕기 위해서 작성되었습니다.

```
# 1. 토큰화
tokenized_text = [['Hope', 'to', 'see', 'you', 'soon'], ['Nice', 'to', 'see', 'you', 'again']]

# 2. 각 단어에 대한 정수 인코딩
encoded_text = [[0, 1, 2, 3, 4],[5, 1, 2, 3, 6]]

# 3. 위 정수 인코딩 데이터가 아래의 임베딩 층의 입력이 된다.
vocab_size = 7
embedding_dim = 2
Embedding(vocab_size, embedding_dim, input_length=5)

# 각 정수는 아래의 테이블의 인덱스로 사용되며 Embedding()은 각 단어마다 임베딩 벡터를 리턴한다.
+------------+------------+
|   index    | embedding  |
+------------+------------+
|     0      | [1.2, 3.1] |
|     1      | [0.1, 4.2] |
|     2      | [1.0, 3.1] |
|     3      | [0.3, 2.1] |
|     4      | [2.2, 1.4] |
|     5      | [0.7, 1.7] |
|     6      | [4.1, 2.0] |
+------------+------------+

# 위의 표는 임베딩 벡터가 된 결과를 예로서 정리한 것이고 Embedding()의 출력인 3D 텐서를 보여주는 것이 아님.
```

Embedding()의 대표적인 인자는 다음과 같습니다.

첫번째 인자 = 단어 집합의 크기. 즉, 총 단어의 개수
두번째 인자 = 임베딩 벡터의 출력 차원. 결과로서 나오는 임베딩 벡터의 크기
input_length = 입력 시퀀스의 길이

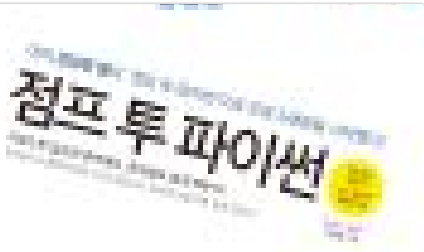
LSTM 구조 자체가 있기에 bias가 포함 안 된다.

```
lstm (LSTM)                  (None, 8)                 800
```

점프 투 파이썬

이 책은 파이썬이란 언어를 처음 접해보는 독자들과 프로그래밍을 한 번도 해 본적이 없는 사람들을 대상으로 한다. 프로그래밍을 할 때 사용되는 전문적인 용어들을 알기 쉽게 풀어서 ...

https://wikidocs.net/32105



02_IMDB_LSTM

[1]:
```
# IMDB 리뷰 데이터 셋
from tensorflow.keras.datasets import imdb

(train_input, train_target),(test_input, test_target) = imdb.load_data(num_words=500) # skip_top=20
# num_words=500는 500순위까지만 끊었음
# skip_top - 1~20위까지는 생략
# 500순위 단어를 벗어나는 것은 2로 채움.
```

[2]:
```
print(train_input.shape, test_input.shape) # 데이터의 의미를 정확하게 파악
```

```
(25000,) (25000,)
```

[3]:
```
print(len(train_input[0]), len(train_input[1])) # 첫번째 댓글은 218단어, 두번째 189단어
```

```
218 189
```

[4]:
```
print(train_input[0]) # 단어 빈도수 순위(top 20을 배제한)
```

```
[1, 14, 22, 16, 43, 2, 2, 2, 2, 65, 458, 2, 66, 2, 4, 173, 36, 256, 5, 25, 100, 43, 2, 112, 50, 2, 2, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 2, 336, 385, 39, 4, 172, 2, 2, 17, 2, 38, 13, 447, 4, 192, 50, 16, 6, 147, 2, 19, 14, 22, 4, 2, 2, 469, 4, 22, 71, 87, 12, 16, 43, 2, 38, 76, 15, 13, 2, 4, 22, 17, 2, 17, 12, 16, 2, 18, 2, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 2, 2, 16, 480, 66, 2, 33, 4, 130, 12, 16, 38, 2, 5, 25, 124, 51, 36, 135, 48, 25, 2, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 2, 8, 4, 107, 117, 2, 15, 256, 4, 2, 7, 2, 5, 2, 36, 71, 43, 2, 476, 26, 400, 317, 46, 7, 4, 2, 2, 13, 104, 88, 4, 381, 15, 297, 98, 32, 2, 56, 26, 141, 6, 194, 2, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 2, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 2, 88, 12, 16, 283, 5, 16, 2, 113, 103, 32, 15, 16, 2, 19, 178, 32]
```

[5]:
```
print(train_target[:20]) # 1번은 긍정, 0번은 부정. 그래서 softmax는 필요없게 된다. # 감성 분류
```

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

[6]:
```
# 훈련 세트에서 2(어휘 사전에 없는 단어) 제외하기
for i in range(len(train_input)):
    train_input[i] = [w for w in train_input[i] if w > 2] # 단어 218번만큼 반복

print(train_input[0])
```

```
[14, 22, 16, 43, 65, 458, 66, 4, 173, 36, 256, 5, 25, 100, 43, 112, 50, 9, 35, 480, 284, 5, 150, 4, 172, 112, 167, 336, 385, 39, 4, 172, 17, 38, 13, 447, 4, 192, 50, 16, 6, 147, 19, 14, 22, 4, 469, 4, 22, 71, 87, 12, 16, 43, 38, 76, 15, 13, 4, 22, 17, 17, 12, 16, 18, 5, 62, 386, 12, 8, 316, 8, 106, 5, 4, 16, 480, 66, 33, 4, 130, 12, 16, 38, 5, 25, 124, 51, 36, 135, 48, 25, 33, 6, 22, 12, 215, 28, 77, 52, 5, 14, 407, 16, 82, 8, 4, 107, 117, 15, 256, 4, 7, 5, 36, 71, 43, 476, 26, 400, 317, 46, 7, 4, 13, 104, 88, 4, 381, 15, 297, 98, 32, 56, 26, 141, 6, 194, 18, 4, 226, 22, 21, 134, 476, 26, 480, 5, 144, 30, 18, 51, 36, 28, 224, 92, 25, 104, 4, 226, 65, 16, 38, 88, 12, 16, 283, 5, 16, 113, 103, 32, 15, 16, 19, 178, 32]
```

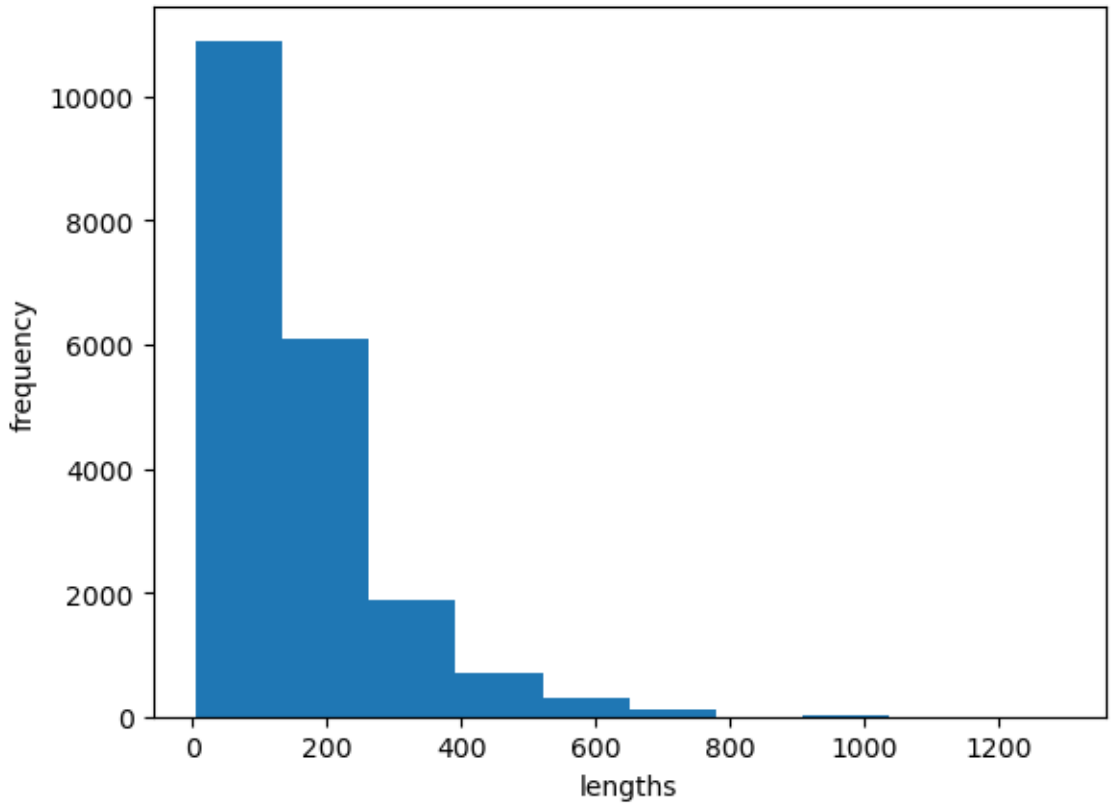
]:
```
from sklearn.model_selection import train_test_split

train_input, val_input, train_target, val_target = #
train_test_split(train_input, train_target, test_size=0.2, random_state=42)
```

]:
```
import numpy as np

lengths = np.array([len(x) for x in train_input])
```

]:
```
print(np.mean(lengths), np.median(lengths))
```

```
164.67985 126.0
```

]:
```
import matplotlib.pyplot as plt

plt.hist(lengths)
plt.xlabel('lengths')
plt.ylabel('frequency')
plt.show()
```



1]:
```
from tensorflow.keras.preprocessing.sequence import pad_sequences

train_seq = pad_sequences(train_input, maxlen=100)
# 100글자 이후만 남기고 나머지 글자는 싹 다 지워버림.
# maxlen - 218개를 100개로 잘라버림
# padding='pre'('post') maxlen에서 자른 것을 0으로 채워줌.
train_seq.shape
```

1]:
```
(20000, 100)
```

2]:
```
train_seq[5]

# 숫자들은 범주형(남자1, 여자2 ...와 같은)
# 49번째 영어단어, 19번째 영어단어라고 함. 유니코드화하여 정수화시킴
#100개의 단어를 가지고 있다..
```

2]:
```
array([  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 195,
        19,  49, 190,   4, 352, 183,  10,  10,  13,  82,  79,   4,  36,
        71, 269,   8,  25,  19,  49,   7,   4,  10,  10,  48,  25,  40,
        11,  40,   5,   4,  95,  14, 238,  56, 129,  10,  10,  21,  94,
       364, 352,  11, 190,  24, 484,   7,  94, 205, 405,  10,  10,  87,
        34,  49,   7, 290,  46,  48,  64,  18,   4])
```

3]:
```
val_seq = pad_sequences(val_input, maxlen=100)
train_seq.shape
```

3]:
```
(20000, 100)
```

순환 신경망 만들기_20221202

```
from tensorflow import keras

model = keras.Sequential()

model.add(keras.layers.Embedding(500, 16, input_length=100))
model.add(keras.layers.LSTM(8))
# 신경망 - Dense, CNN - Conv2D, RNN - SimpleRNN()
# default가 tahn으로 -1~1로 해놔서 달아놓을 필요 없다.
model.add(keras.layers.Dense(1, activation='sigmoid'))

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 100, 16)           8000

 lstm (LSTM)                 (None, 8)                 800

 dense (Dense)               (None, 1)                 9

=================================================================
Total params: 8,809
Trainable params: 8,809
Non-trainable params: 0
_________________________________________________________________
```

```
print(1e-4)
```

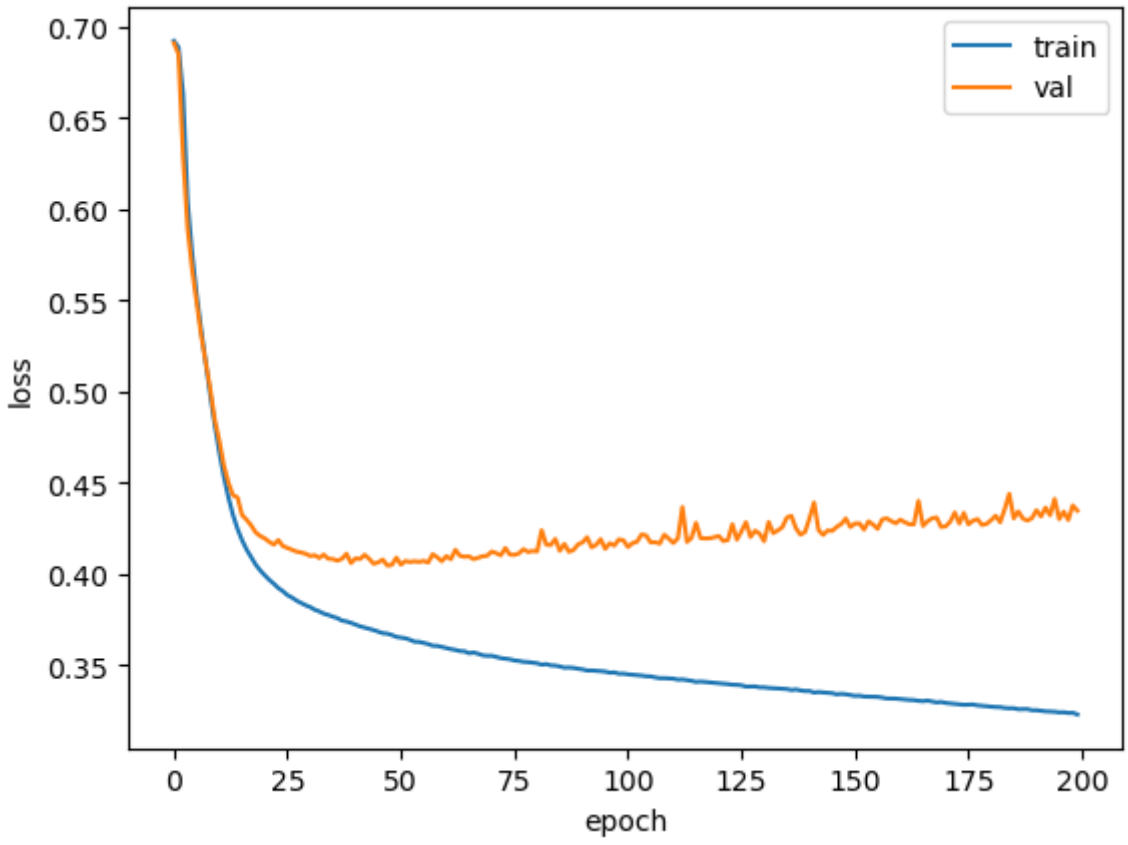
```
0.0001
```

```
model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-4), loss='binary_crossentropy', metrics='accuracy')
# RMSprop은 디폴트값을 정해줄 수 있음.
# learning_rate = 1e-4 = 0.0001

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-LSTM-Model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)

history = model.fit(train_seq, train_target, epochs=200, batch_size=64, validation_data=(val_seq, val_target),
                    callbacks=[checkpoint_cb]) # early_stopping_cb
# batch_size=64 - 64씩 읽어오라는 것.
# batch_size만큼 가져와서 validation_data로 검증도 같이함.
# history - 정확도 값을 담아줌
# patience=3 - 3번이상 커지면 멈춰라! # patience의 기준은 val_loss
```

순환 신경망은 시계열 데이터에서 사용 가능하다. text에만 적용 가능한 것이 아님.

```
Epoch 46/200
313/313 [==============================] - 7s 21ms/step - loss: 0.3684 - accuracy: 0.8400 - val_loss: 0.4063 - val_accuracy: 0.8176
Epoch 47/200
313/313 [==============================] - 6s 21ms/step - loss: 0.3675 - accuracy: 0.8403 - val_loss: 0.4079 - val_accuracy: 0.8164
Epoch 48/200
313/313 [==============================] - 7s 21ms/step - loss: 0.3673 - accuracy: 0.8418 - val_loss: 0.4046 - val_accuracy: 0.8176
Epoch 49/200
313/313 [==============================] - 7s 21ms/step - loss: 0.3665 - accuracy: 0.8410 - val_loss: 0.4049 - val_accuracy: 0.8152
Epoch 50/200
313/313 [==============================] - 6s 21ms/step - loss: 0.3655 - accuracy: 0.8416 - val_loss: 0.4089 - val_accuracy: 0.8144
Epoch 51/200
313/313 [==============================] - 8s 24ms/step - loss: 0.3651 - accuracy: 0.8407 - val_loss: 0.4051 - val_accuracy: 0.8188
Epoch 52/200
313/313 [==============================] - 7s 22ms/step - loss: 0.3647 - accuracy: 0.8410 - val_loss: 0.4071 - val_accuracy: 0.8174
Epoch 53/200
313/313 [==============================] - 8s 25ms/step - loss: 0.3639 - accuracy: 0.8422 - val_loss: 0.4065 - val_accuracy: 0.8166
Epoch 54/200
313/313 [==============================] - 7s 23ms/step - loss: 0.3628 - accuracy: 0.8429 - val_loss: 0.4069 - val_accuracy: 0.8168
Epoch 55/200
```

```
"""
Epoch 51/200
313/313 [==============================] - 8s 24ms/step - loss: 0.3651 - accuracy: 0.8407 - val_loss: 0.4051 - val_accuracy: 0.8188
"""

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('epoch')
plt.ylabel('loss') # 두 개의 loss를 출력해줌. loss, val_loss
plt.legend(['train', 'val']) # 색깔로 표시해줌.
plt.show()
```



단어 임베딩 사용하기_20221202

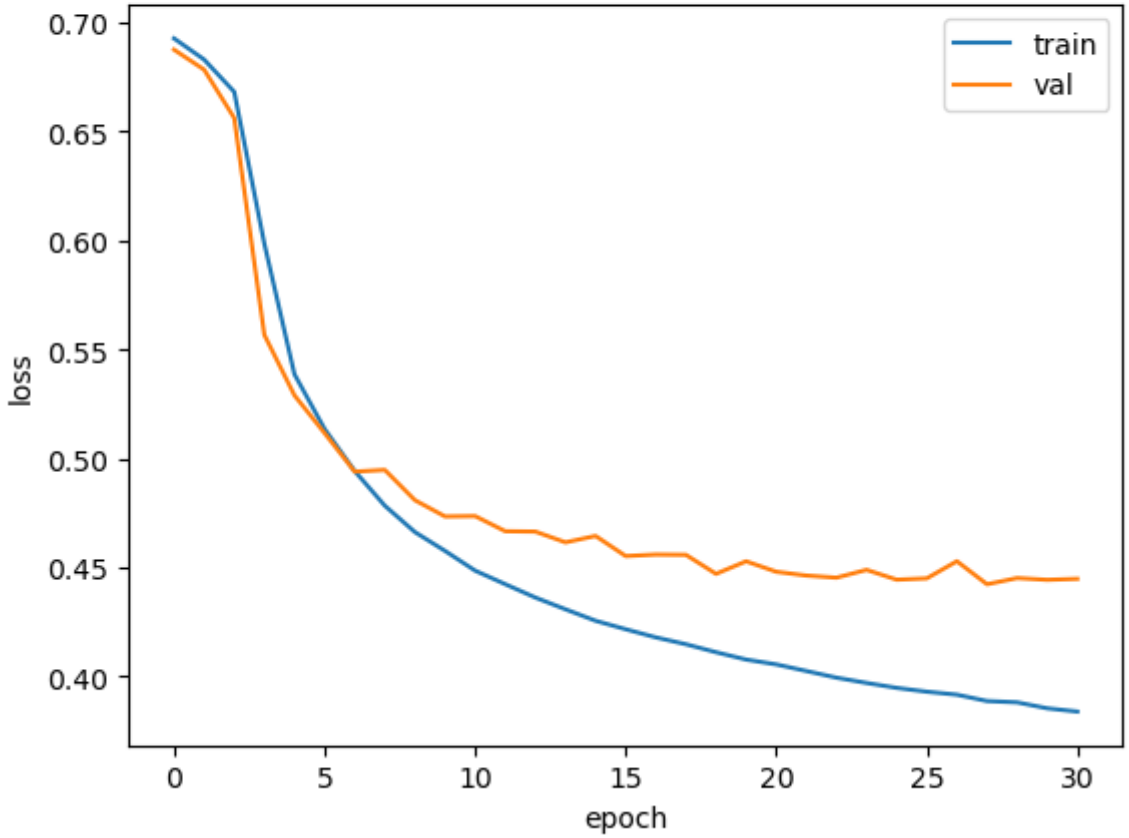
8]:
```
model2 = keras.Sequential()

model2.add(keras.layers.Embedding(500, 16, input_length=100))
# 500개의 one-hot을 16개로 줄여줌.
model2.add(keras.layers.SimpleRNN(8)) # SimpleRNN을 LSTM으로만 바꿔주면 앞으로의 진도는 거의 끝
model2.add(keras.layers.Dense(1, activation='sigmoid'))
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 100, 16)           8000

 simple_rnn (SimpleRNN)      (None, 8)                 200

 dense_1 (Dense)             (None, 1)                 9

=================================================================
Total params: 8,209
Trainable params: 8,209
Non-trainable params: 0
_________________________________________________________________
```

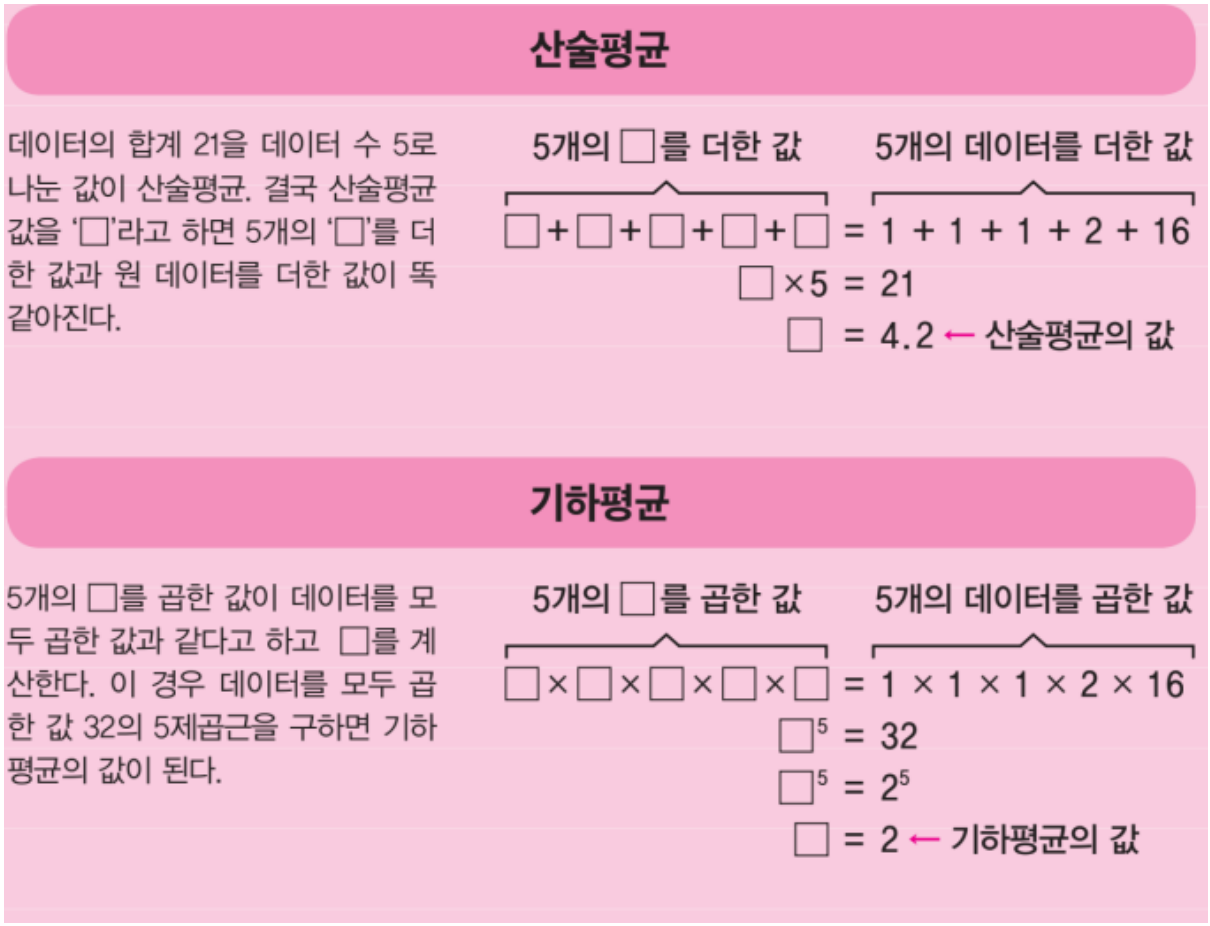
```
model2.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1e-4),
               loss='binary_crossentropy', metrics='accuracy')
# RMSprop은 디폴트값을 정해줄 수 있음.
# learning_rate = 1e-4 = 0.0001

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-embeddingRNN-Model.h5', save_best_only=True)
early_stopping_cb = keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True)

history = model2.fit(train_seq, train_target, epochs=50, batch_size=64,
                     validation_data=(val_seq, val_target),
                     callbacks=[checkpoint_cb, early_stopping_cb])
```

```
313/313 [==============================] - 3s 11ms/step - loss: 0.4025 - accuracy: 0.8304 - val_loss: 0.4462 - val_accuracy: 0.7964
Epoch 23/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3994 - accuracy: 0.8334 - val_loss: 0.4453 - val_accuracy: 0.7970
Epoch 24/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3970 - accuracy: 0.8327 - val_loss: 0.4489 - val_accuracy: 0.7940
Epoch 25/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3947 - accuracy: 0.8343 - val_loss: 0.4444 - val_accuracy: 0.7968
Epoch 26/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3930 - accuracy: 0.8357 - val_loss: 0.4450 - val_accuracy: 0.7956
Epoch 27/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3916 - accuracy: 0.8362 - val_loss: 0.4528 - val_accuracy: 0.7956
Epoch 28/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3887 - accuracy: 0.8378 - val_loss: 0.4423 - val_accuracy: 0.7990
Epoch 29/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3881 - accuracy: 0.8379 - val_loss: 0.4451 - val_accuracy: 0.7980
Epoch 30/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3854 - accuracy: 0.8396 - val_loss: 0.4443 - val_accuracy: 0.7970
Epoch 31/50
313/313 [==============================] - 3s 11ms/step - loss: 0.3839 - accuracy: 0.8397 - val_loss: 0.4447 - val_accuracy: 0.7956
```

```
]: plt.plot(history.history['loss'])
   plt.plot(history.history['val_loss'])
   plt.xlabel('epoch')
   plt.ylabel('loss') # 두 개의 loss를 출력해줌. loss, val_loss
   plt.legend(['train', 'val']) # 색깔로 표시해줌.
   plt.show()
```



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/1b4edf1e-51aa-412a-a9cc-7748b9bdd86d/02_IMDB_LSTM_20221202.ipynb



03_Stock_LSTM_PredictionModel

1]:
```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

2]:
```
def MinMaxScaler(data):
    numerator = data - np.min(data, axis=0)
    denominator = np.max(data, axis=0) - np.min(data, axis=0)
    return numerator / (denominator + 1e-7)
```

```
xy = np.loadtxt('data/data-02-stock_daily.csv', delimiter=',') # 주식 데이터
xy
```

```
array([[8.28659973e+02, 8.33450012e+02, 8.28349976e+02, 1.24770000e+06,
        8.31659973e+02],
       [8.23020020e+02, 8.28070007e+02, 8.21655029e+02, 1.59780000e+06,
        8.28070007e+02],
       [8.19929993e+02, 8.24400024e+02, 8.18979980e+02, 1.28170000e+06,
        8.24159973e+02],
       ...,
       [5.66892592e+02, 5.67002574e+02, 5.56932537e+02, 1.08000000e+04,
        5.56972503e+02],
       [5.61202549e+02, 5.66432590e+02, 5.58672539e+02, 4.12000000e+04,
        5.59992565e+02],
       [5.68002570e+02, 5.68002570e+02, 5.52922516e+02, 1.31000000e+04,
        5.58462551e+02]])
```

```
xy = xy[::-1] # 순환망은 순서가 의미를 가지는 데이터셋이다. # -1로 순서를 뒤집어줌.
xy
```

```
array([[5.68002570e+02, 5.68002570e+02, 5.52922516e+02, 1.31000000e+04,
        5.58462551e+02],
       [5.61202549e+02, 5.66432590e+02, 5.58672539e+02, 4.12000000e+04,
        5.59992565e+02],
       [5.66892592e+02, 5.67002574e+02, 5.56932537e+02, 1.08000000e+04,
        5.56972503e+02],
       ...,
       [8.19929993e+02, 8.24400024e+02, 8.18979980e+02, 1.28170000e+06,
        8.24159973e+02],
       [8.23020020e+02, 8.28070007e+02, 8.21655029e+02, 1.59780000e+06,
        8.28070007e+02],
       [8.28659973e+02, 8.33450012e+02, 8.28349976e+02, 1.24770000e+06,
        8.31659973e+02]])
```

```
total_size = int(len(xy))
total_size
```

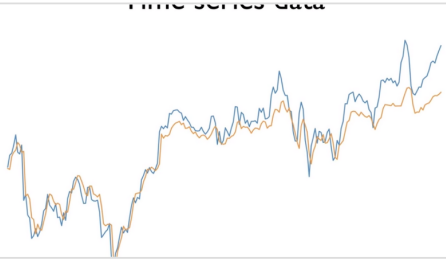
```
732
```

참고 사이트

RNN with Time Series Data

RNN을 이용하여 Stock Marcket과 같은 time series를 예측하는 방법에 대해 이야기 해보겠다.¶ time series data란 무엇일까? 어떤 시간에 따라 변화하는 값을 의미한다. x축은 시간 y축은 value가 될 것이

https://jfun.tistory.com/194



```
# train Parameters
seq_length = 7
input_feature = 5

train_size = int(len(xy) * 0.7)
train_set = xy[0:train_size]
test_set = xy[train_size - seq_length:]
```

```
print(train_set.shape, test_set.shape)
```

```
(512, 5) (227, 5)
```

```
train_set = MinMaxScaler(train_set)
test_set = MinMaxScaler(test_set)
```

```
def build_dataset(time_series, seq_length):
    dataX = []
    dataY = []

    for i in range(0, len(time_series) - seq_length):
        x = time_series[i:i+seq_length, :]
        y = time_series[i+seq_length, [-1]]
        print(x, ' -> ', y)
        dataX.append(x)
        dataY.append(y)

    return np.array(dataX), np.array(dataY)
```

```
trainX, trainY = build_dataset(train_set, seq_length)
testX, testY = build_dataset(test_set, seq_length)
print(trainX.shape, trainY.shape)
```

```
 [0.84347469 0.8426171  0.88749985 0.10121322 0.86858608]
 [0.86925245 0.87626864 0.92209184 0.1140722  0.90185774]
 [0.88723699 0.88829938 0.92518158 0.08714288 0.90908564]
 [0.88939504 0.88829938 0.94014512 0.13380794 0.90030461]]  ->  [0.93124657]
[[0.81529885 0.82251706 0.84757626 0.1060959  0.83698714]
 [0.83034597 0.81556125 0.85575466 0.07523435 0.84403567]
 [0.84347469 0.8426171  0.88749985 0.10121322 0.86858608]
 [0.86925245 0.87626864 0.92209184 0.1140722  0.90185774]
 [0.88723699 0.88829938 0.92518158 0.08714288 0.90908564]
 [0.88939504 0.88829938 0.94014512 0.13380794 0.90030461]
 [0.89281215 0.89655181 0.94323484 0.12965206 0.93124657]]  ->  [0.95460261]
[[0.83034597 0.81556125 0.85575466 0.07523435 0.84403567]
 [0.84347469 0.8426171  0.88749985 0.10121322 0.86858608]
 [0.86925245 0.87626864 0.92209184 0.1140722  0.90185774]
 [0.88723699 0.88829938 0.92518158 0.08714288 0.90908564]
 [0.88939504 0.88829938 0.94014512 0.13380794 0.90030461]
 [0.89281215 0.89655181 0.94323484 0.12965206 0.93124657]
 [0.91133638 0.91818448 0.95944078 0.1885611  0.95460261]]  ->  [0.97604677]
(505, 7, 5) (505, 1)
```

```
print(trainX.shape, trainY.shape)
```

```
(505, 7, 5) (505, 1)
```

```
model = tf.keras.Sequential()
model.add(tf.keras.layers.LSTM(units=1, input_shape=(seq_length, input_feature)))
model.add(tf.keras.layers.Dense(units=1, activation='tanh'))
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_2 (LSTM)               (None, 1)                 28

 dense_2 (Dense)             (None, 1)                 2

=================================================================
Total params: 30
Trainable params: 30
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(learning_rate=0.01))
model.fit(trainX, trainY, epochs=100)
```

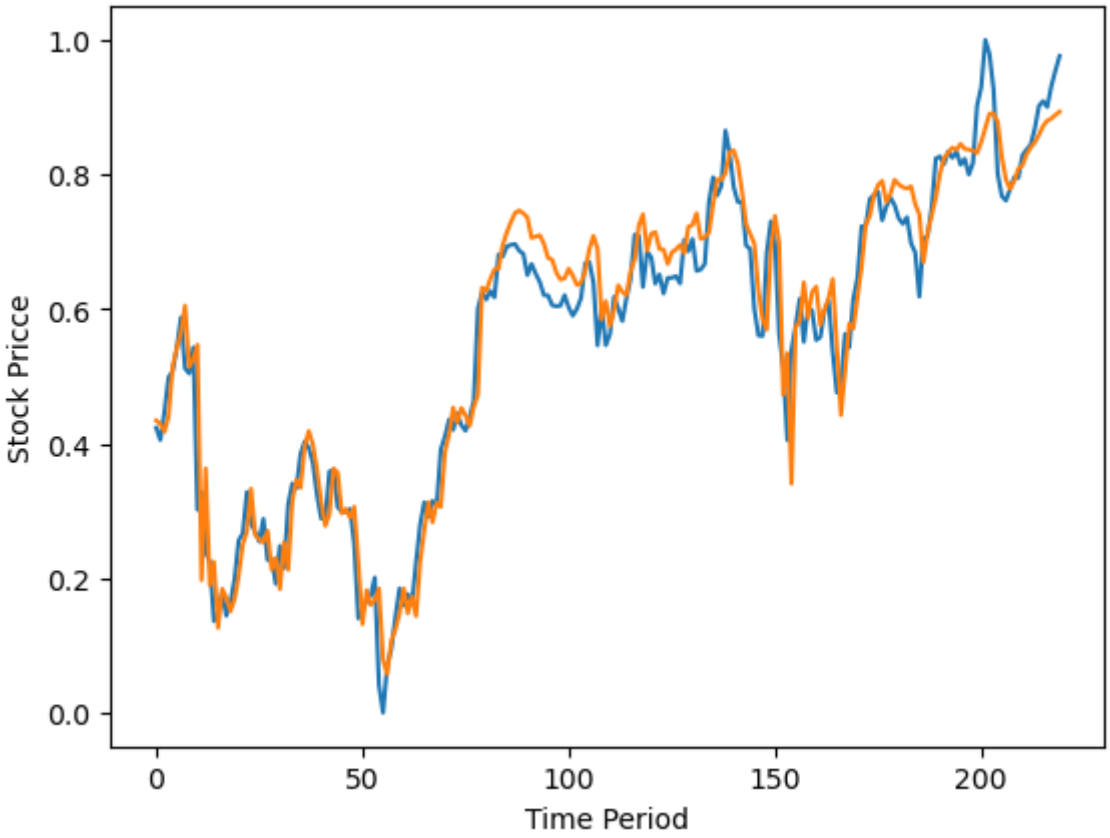
```
Epoch 92/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0014
Epoch 93/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0015
Epoch 94/100
16/16 [==============================] - 0s 2ms/step - loss: 0.0015
Epoch 95/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0014
Epoch 96/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0014
Epoch 97/100
16/16 [==============================] - 0s 2ms/step - loss: 0.0014
Epoch 98/100
16/16 [==============================] - 0s 2ms/step - loss: 0.0014
Epoch 99/100
16/16 [==============================] - 0s 3ms/step - loss: 0.0014
Epoch 100/100
16/16 [==============================] - 0s 2ms/step - loss: 0.0014
```

```
<keras.callbacks.History at 0x1f189bb1488>
```

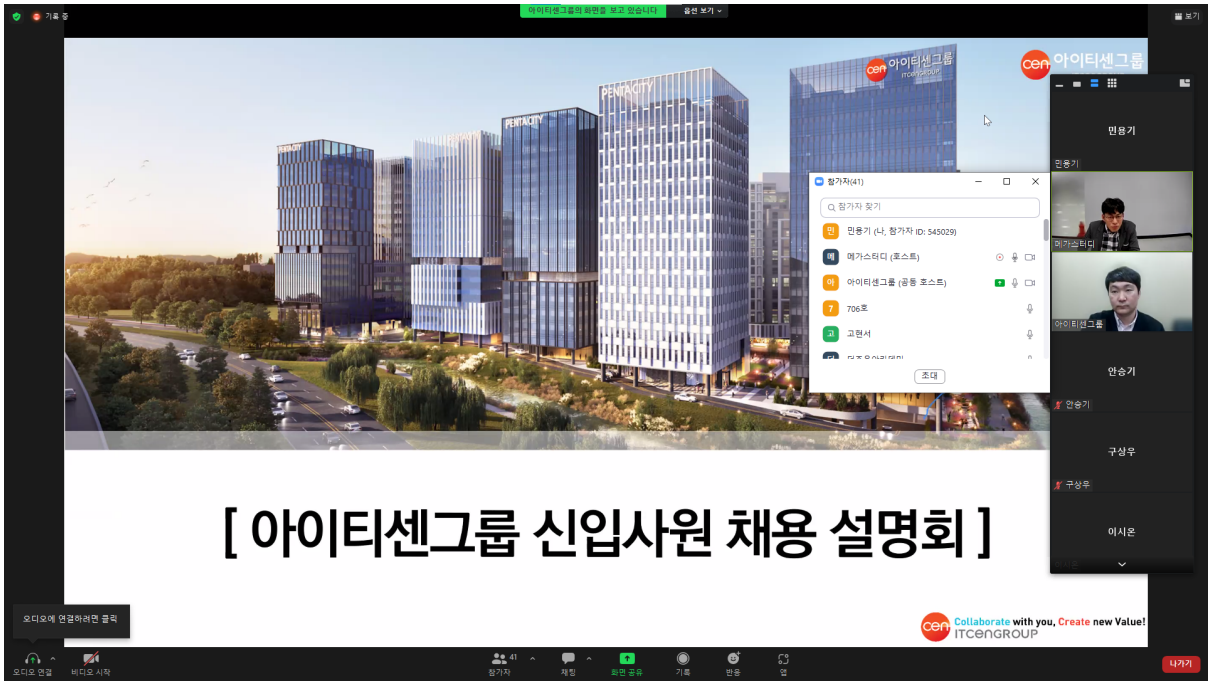
```
test_predict = model.predict(testX)
print(testX.shape, test_predict.shape)
```

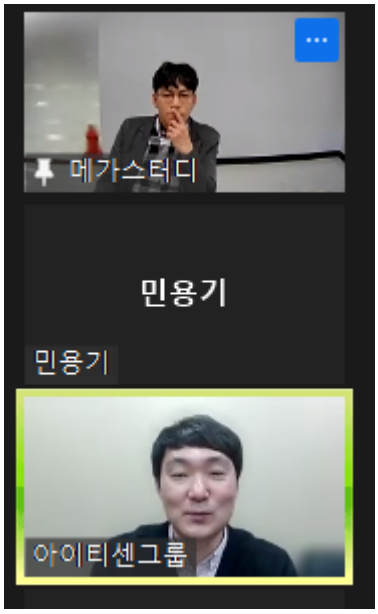
```
7/7 [==============================] - 0s 1ms/step
(220, 7, 5) (220, 1)
```

```
import os
os.environ['KMP_DUPLICATE_LIB_OK'] = 'True'

plt.plot(testY)
plt.plot(test_predict)
plt.xlabel('Time Period')
plt.ylabel('Stock Pricce')
plt.legend(['target','predict'])
plt.show()
```



Zoom 아이티센그룹 채용 설명회





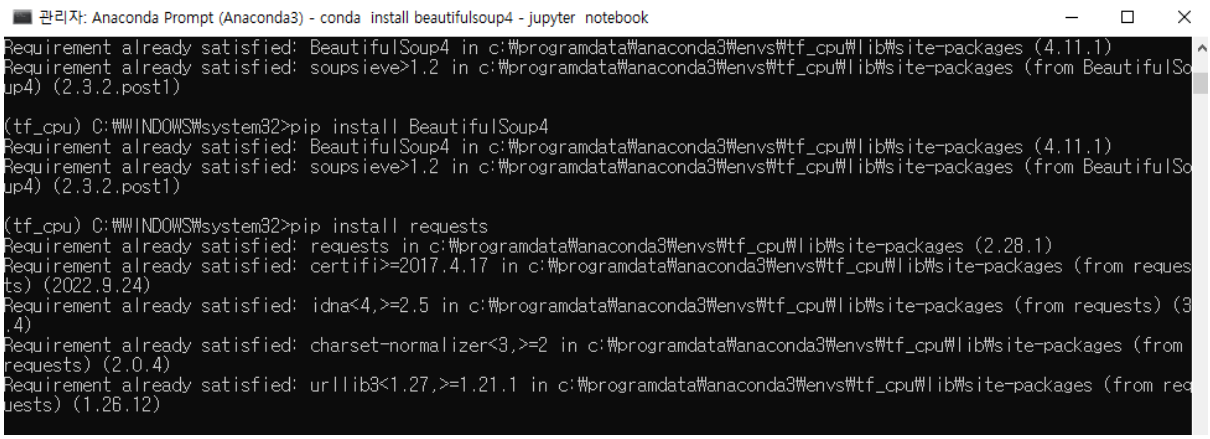
- 매출 3조
- 2005년 시작

현재 대중교통 요금

구분	전철(카드)	전철(현금)	시내버스(카드)	시내버스(현금)	택시(중형)
서울	저 1,250	저 1,350	1,200	1,300	3,800

01_robots_readme

패키지 설치



다시 jupyter notebook 접속

```
: # 웹 크롤링을 위한 패키지 설치
  # pip install BeautifulSoup4
  # pip intall requests

  import requests
  from bs4 import BeautifulSoup
```

```
: urls = ['https://www.naver.com/','https://www.python.org/']
  filename = "robots.txt"

  for url in urls:
      file_path = url + filename
      print(file_path)
      response = requests.get(file_path)
      print(response.text, '₩n')
```

```
https://www.naver.com/robots.txt
User-agent: *
Disallow: /
Allow : /$


https://www.python.org/robots.txt
# Directions for robots.  See this URL:
# http://www.robotstxt.org/robotstxt.html
# for a description of the file format.

User-agent: HTTrack
User-agent: puf
User-agent: MSIECrawler
Disallow: /

# The Krugle web crawler (though based on Nutch) is OK.
User-agent: Krugle
Allow: /
Disallow: /~guido/orlijn/
Disallow: /webstats/

# No one should be crawling us with Nutch.
User-agent: Nutch
Disallow: /

# Hide old versions of the documentation and various large sets of files.
User-agent: *
Disallow: /~guido/orlijn/
Disallow: /webstats/
```

`User-agent: *` 모두

`Disallow` 전혀 허용하지 않겠다. = 차단하겠다.

`Allow` 단, /$ - 첫 메인페이지는 접근 허용한다는 것을 의미.

아래의 로봇들은 차단하겠다.

```
User-agent: HTTrack
User-agent: puf
User-agent: MSIECrawler
Disallow: /
```

Krugle은 허용하지만 허용하지 않는 인폼들(/~guido/orlijn/, /webstats/) 을 따로 구별해놓음.

```
# The Krugle web crawler (though based on Nutch) is OK.
User-agent: Krugle
Allow: /
Disallow: /~guido/orlijn/
Disallow: /webstats/
```

Nutch는 허용 안 하겠다. = 차단하겠다.

`No one should be crawling us with Nutch.`

```
User-agent: Nutch
Disallow: /
```

02_BeautifulSoup

```
]: import requests
   from bs4 import BeautifulSoup
```

```
2]: url = "https://en.wikipedia.org/wiki/Seoul_Metropolitan_Subway"
    resp = requests.get(url)
    html_src = resp.text
    html_src
```

```
]: soup = BeautifulSoup(html_src, 'html.parser')
   # html로 parsing 될 수 있게끔 준비를 하라고 지시함.
   print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

print(soup.head)

```
print(soup.head)
```

```
180916-103548.jpg" property="og:image"/>
<meta content="640" property="og:image:width"/>
<meta content="360" property="og:image:height"/>
<meta content="width=1000" name="viewport"/>
<meta content="Seoul Metropolitan Subway - Wikipedia" property="og:title"/>
<meta content="website" property="og:type"/>
<link href="//upload.wikimedia.org" rel="preconnect"/>
<link href="//en.m.wikipedia.org/wiki/Seoul_Metropolitan_Subway" media="only screen and (max-width: 720px)" rel="alternate"/>
<link href="/w/index.php?title=Seoul_Metropolitan_Subway&amp;action=edit" rel="alternate" title="Edit this page" type="application/x-wi
ki"/>
<link href="/static/apple-touch/wikipedia.png" rel="apple-touch-icon"/>
<link href="/static/favicon/wikipedia.ico" rel="icon"/>
<link href="/w/opensearch_desc.php" rel="search" title="Wikipedia (en)" type="application/opensearchdescription+xml"/>
<link href="//en.wikipedia.org/w/api.php?action=rsd" rel="EditURI" type="application/rsd+xml"/>
<link href="https://creativecommons.org/licenses/by-sa/3.0/" rel="license"/>
<link href="https://en.wikipedia.org/wiki/Seoul_Metropolitan_Subway" rel="canonical"/>
<link href="//meta.wikimedia.org" rel="dns-prefetch"/>
<link href="//login.wikimedia.org" rel="dns-prefetch"/>
</head>
```

print(soup.body)

```
print(soup.body)
```

```
44 Template:Cite_web"," 13.82%  147.390      1 Template:Infobox_Korean_name"," 11.22%  119.578      1 Template:Infobox_Chinese/Korea
n"," 10.81%  115.301      1 Template:Public_transport_in_the_Seoul_Metropolitan_Area","  9.94%  106.011      2 Template:Lang","  8.10%
86.307      2 Template:Ambox","  7.78%   82.979      1 Template:Infobox_public_transit"]},"scribunto":{"limitreport-timeusage":{"valu
e":"0.502","limit":"10.000"},"limitreport-memusage":{"value":35419590,"limit":52428800}},"cachereport":{"origin":"mw2384","timestam
p":"20221130134458","ttl":1814400,"transientcontent":false}}});});</script>
<script type="application/ld+json">{"@context":"https:\/\/schema.org","@type":"Article","name":"Seoul Metropolitan Subway","url":"http
s:\/\/en.wikipedia.org\/wiki\/Seoul_Metropolitan_Subway","sameAs":"http:\/\/www.wikidata.org\/entity\/Q16950","mainEntity":"http:\/\/ww
w.wikidata.org\/entity\/Q16950","author":{"@type":"Organization","name":"Contributors to Wikimedia projects"},"publisher":{"@type":"Org
anization","name":"Wikimedia Foundation, Inc.","logo":{"@type":"ImageObject","url":"https:\/\/www.wikimedia.org\/static\/images\/wmf-ho
r-googpub.png"}},"datePublished":"2003-08-11T08:41:49Z","dateModified":"2022-11-06T09:34:53Z","image":"https:\/\/upload.wikimedia.org\/
wikipedia\/commons\/b\/b4\/Seoul-metro-2009-20180916-103548.jpg","headline":"Seoul metropolitan railway system"}</script><script type
="application/ld+json">{"@context":"https:\/\/schema.org","@type":"Article","name":"Seoul Metropolitan Subway","url":"https:\/\/en.wiki
pedia.org\/wiki\/Seoul_Metropolitan_Subway","sameAs":"http:\/\/www.wikidata.org\/entity\/Q16950","mainEntity":"http:\/\/www.wikidata.or
g\/entity\/Q16950","author":{"@type":"Organization","name":"Contributors to Wikimedia projects"},"publisher":{"@type":"Organization","n
ame":"Wikimedia Foundation, Inc.","logo":{"@type":"ImageObject","url":"https:\/\/www.wikimedia.org\/static\/images\/wmf-hor-googpub.pn
g"}},"datePublished":"2003-08-11T08:41:49Z","dateModified":"2022-11-06T09:34:53Z","image":"https:\/\/upload.wikimedia.org\/wikipedia\/c
ommons\/b\/b4\/Seoul-metro-2009-20180916-103548.jpg","headline":"Seoul metropolitan railway system"}</script>
<script>(RLQ=window.RLQ||[]).push(function(){mw.config.set({"wgBackendResponseTime":123,"wgHostname":"mw2275"});});</script>
</body>
```

```
print('title 태그 요소: ', soup.title)
print('title 태그 이름: ', soup.title.name)
print('title 태그 문자열: ', soup.title.string) # 문자열만 반환해줌
```

```
title 태그 요소:  <title>Seoul Metropolitan Subway - Wikipedia</title>
title 태그 이름:  title
title 태그 문자열:  Seoul Metropolitan Subway - Wikipedia
```

first_img = soup.find(name='img')
first_img

```
first_img = soup.find(name='img')
first_img
```

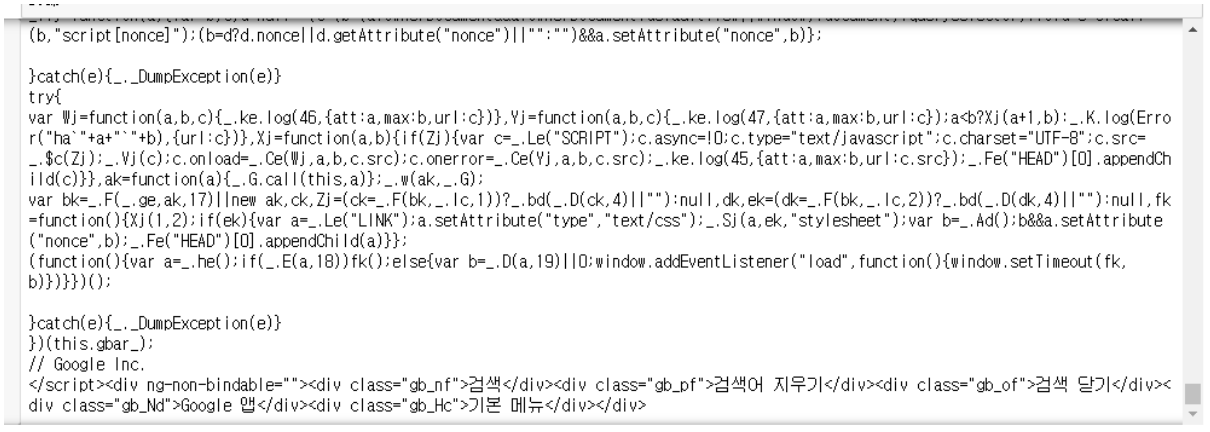
```
<img alt="" data-file-height="40" data-file-width="40" decoding="async" height="40" src="//upload.wikimedia.org/wikipedia/en/thumb/b/b4/Ambox_important.svg/40px-Ambox_important.svg.png" srcset="//upload.wikimedia.org/wikipedia/en/thumb/b/b4/Ambox_important.svg/60px-Ambox_important.svg.png 1.5x, //upload.wikimedia.org/wikipedia/en/thumb/b/b4/Ambox_important.svg/80px-Ambox_important.svg.png 2x" width="40"/>
```

```
target_img = soup.find(name='img', attrs={'alt':'South Korea subway logo.svg'})
target_img
```

```
<img alt="South Korea subway logo.svg" data-file-height="450" data-file-width="450" decoding="async" height="75" src="//upload.wikimedia.org/wikipedia/commons/thumb/1/12/South_Korea_subway_logo.svg/75px-South_Korea_subway_logo.svg.png" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/1/12/South_Korea_subway_logo.svg/113px-South_Korea_subway_logo.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/1/12/South_Korea_subway_logo.svg/150px-South_Korea_subway_logo.svg.png 2x" width="75"/>
```

03_News_Parsing

```
# 구글 뉴스 검색(검색어: 파이썬)
base_url = "https://news.google.com"
search_url = base_url + "/search?q=%ED%8C%8C%EC%9D%B4%EC%8D%AC&hl=ko&gl=KR&ceid=KR%3Ako"
resp = requests.get(search_url)
html_src = resp.text
soup = BeautifulSoup(html_src, 'html.parser')
soup
```


(b,"script[nonce]");(b=d?d.nonce||d.getAttribute("nonce")||"":"")&&a.setAttribute("nonce",b)};
}catch(e){_._DumpException(e)}
try{
var Wj=function(a,b,c){_.ke.log(46,{att:a,max:b,url:c})},Yj=function(a,b,c){_.ke.log(47,{att:a,max:b,url:c});a<b?Xj(a+1,b):_.K.log(Error("ha`"+a+"``+b),{url:c})},Xj=function(a,b){if(Zj){var c=_.Le("SCRIPT");c.async=!0;c.type="text/javascript";c.charset="UTF-8";c.src=_.$c(Zj);_.Vj(c);c.onload=_.Ce(Wj,a,b,c.src);c.onerror=_.Ce(Yj,a,b,c.src);_.ke.log(45,{att:a,max:b,url:c.src});_.Fe("HEAD")[0].appendChild(c)}},ak=function(a){_.G.call(this,a)};_.w(ak,_.G);
var bk=_.F(_.ge,ak,17)||new ak,ck,Zj=(ck=_.F(bk,_.lc,1))?_.bd(_.D(ck,4)||""):null,dk,ek=(dk=_.F(bk,_.lc,2))?_.bd(_.D(dk,4)||""):null,fk=function(){Xj(1,2);if(ek){var a=_.Le("LINK");a.setAttribute("type","text/css");_.Sj(a,ek,"stylesheet");var b=_.Ad();b&&a.setAttribute("nonce",b);_.Fe("HEAD")[0].appendChild(a)}};
(function(){var a=_.he();if(_.E(a,18))fk();else{var b=_.D(a,19)||0;window.addEventListener("load",function(){window.setTimeout(fk,b)})}})();
}catch(e){_._DumpException(e)}
})(this.gbar_);
// Google Inc.
</script> 검색 검색어 지우기 검색 닫기 Google 앱 기본 메뉴

```
# 뉴스 아이템 블록을 선택
# DY5T1d RZIKme
news_items = soup.select('a[class="DY5T1d RZIKme"]')
print(len(news_items))
print(news_items[0])
```

```
100
<a class="DY5T1d RZIKme" href="./articles/CBMiQWh0dHBz0i
aHR0cHM6Ly93d3cuY29kaW5nd29ybGRuZXdzLmNvbS9uZXdzL2FydGlj
해야 할 10가지 파이썬 개발 트렌드</a>
```

```
# xrnccd
news_items = soup.select('div[class="xrnccd"]')
print(len(news_items))
print(news_items[0])
```


```
GwsbnVsbCxudWxsLG51bGwsbnVsbCxudWxsLG51bGwsbnVsbCxudWxsLG51bGwsbnVsbCxudWxsLG51bGwsbr
NDg1NjA4IiwiMTEzNTMwNDc3MjQ0ODk4NTY5NTkiLCJodHRwczovL3d3dy5jb2Rpb
nlixbMzFdXSxbW1tudWxsLCLtjlzsnbTsjawiXVOsMVOsbnVsbCwyLG51bGwsbnVs
AwMF1dXQ==" jsmodel="tOLjce hECoeb"><div aria-disabled="false" ar
d V3dfMc wOhkKb M9Bg4d" data-n-et="1000" data-tooltip="공유" data
tion="click:cOuCgd; mousedown:UX7yZ; mouseup:lbsD7e; mouseenter:t
f;touchstart:p6p2H; touchmove:FwuNnf; touchend:yfqBxc(preventMous
Sufsc:BS8cLb;RyWlBb:tC9Erd;UTnG9:aDaYxb;nUyoxf:El6wk;" jscontroll
v class="Xl1LOd zSBur" jsname="ksKsZd"></div><span aria-hidden="t
</div></span><span class="L8PZAb GB1Zid" data-n-prms="[false,fals
6b:npT2md;TDui6d:BgNF9e;aWRkAb:uOWEMd;h4C2te:Oy8cwd;" jscontrolle
b2OvbmV3cy9hcnRpY2xlVmlldy5odG1sP2lkeG5vPTEzNzIyOgFEaHR0cHM6Ly93c
9MTM3MjI;4" jslog="109017" jsmodel="WDTLsd BZ12ub"><div aria-disa
보기" class="U26fgb YOnsCc waNn5b ZqhUjb ztUP4e uUmlDd gL67me cd2
ta-tooltip-horizontal-offset="0" data-tooltip-vertical-offset="-1
r:tf01Yc; mouseleave:JywGue; focus:AHmuwe; blur:O22p3e; contextme
ouseEvents=true|preventDefault=true); touchcancel:JMtRjd;;keydowr
El6wk;" jscontroller="S9Bhuc" jsname="itaskb" jsshadow="" role="b
pan aria-hidden="true" class="DPvwYc ChwdAb XdO67b fAk9Qc" jsname
```