LSTM과 CNN의 혼합 모델을 사용한 주가 데이터 예측 가이드

목차

목표

본 논문의 첫번째 목표

CNN과 LSTM의 다양한 모델 조합으로 일주일 뒤 혹은 한달 뒤 주가의 상승, 하락을
예측하는 좋은 모델을 선별하는 것입니다.
또한 추가로 강한 상승, 약한 상승, 약한 하락, 강한 하락으로 분류하는 경우 모델의 성능을 평가합니다.

본 논문의 두번째 목표

딥러닝을 주가 예측 모델에 적용할 시 정확도를 평가하는 새로운 지표를 나타냅니다.
딥러닝 모델을 사용하여 주가 데이터의 상향 및 하향을 예측하는 모델을 테스트할 시,
대부분의 연구는 Test data set의 정확도를 기반으로 해당 모델이 우수한지 아닌지 판단합니다.

본 논문에서는 주가 데이터의 상승, 하강 분류 모델에 대한 테스트 시
Test data set의 정확도만을 지표로 할 때의 문제점을 제시함과 더불어,
우수한 모델을 고르는 정확한 지표에 대한 기준점을 제시합니다.

주요 논점

1. 전체 학습데이터 및 배치데이터

week data (X) : x+40 일 간의 데이터
week data (Y) : x+40일 데이터의 5일 뒤 상승폭 하락폭 학습 -> 1. week data set 구성

month data (X) : x+40 일 간의 데이터
month data (Y) : x+40일 데이터의 20일 뒤상승폭 하락폭 학습 -> 2. month data set 구성

2. 상승 하락 분류 방법

2가지의 상승 하락 체크 방법
4가지의 상승 하락 체크 방법

4가지의 상승 하락 (강한 상승, 약한 상승, 약한 하락, 강한 하락)

강하고 약한 것에 대한 기준점을 찾아야 함 (5% or 10% 등)

주요 논점

3. 데이터의 유형별 특징을 생각하며 테스트를 해봐야함

(우상향, 우하향, 구간 반복 등.. 의 테스트 별 차이 가장 효율적인 데이터 셋은?)

4. 테스트 한 데이터의 전체 정확도 뿐만 아닌, 각 선택 별 정확도를 또한 출력

Ex) 상승을 선택할 때의 정확도
하락을 선택할 때의 정확도

주요 논점

5. 평가 데이터의 학습된 모습

Ex) 우상향하는 주가 그래프에서 학습을 진행할 때

“ 예측된 데이터들이 전부 상승이다 “ -> 잘못된 학습 방법

6. CNN을 사용한 LSTM Model과 단순 LSTM Model의 학습 차이

LSTM의 unit 개수 및 CNN의 필터 개수에 따른 정확도 차이 테스트

주요 논점 (1, 2)

1, 전체 학습데이터 및 배치데이터 구성 2. 상승 하락 체크 방법

정답 데이터의 구분 1 : Binary classification

Binary classification : 50% 이상의 확률

(상승, 하락 체크)

Week data result = 40번째 종가 - 45번째 종가
Month data result = 40번째 종가 - 60번째 종가

Result > 0 : [1, 0] (one - hot encoding)
Result < 0 : [0, 1]

One hot encoding을 사용하여, Data set 구성

주요 논점 (1, 2)

정답 데이터의 구분 2 : 4가지의 분류

4가지의 분류 : 25% 이상의 확률

(강한 상승, 약한 상승, 약한 하락, 강한 하락)

Week data result = 40번째 종가 - 45번째 종가
Month data result = 40번째 종가 - 60번째 종가

$$\frac{\text{한달 전 주가 - 현재가}}{\text{한달 전 주가}} = \text{변화폭 계산 가능}$$

변화 폭을 시각화 -> 평균 변화폭을 살펴봄

Train Data set 구성

주요 논점 (1, 2)

주의) 데이터를 학습 시킬 때 -> Min max scaler 를 사용

정규화된 데이터의 경우 : 제대로 된 정답이 나오지 않음

$$\frac{\text{한달 전 주가 - 현재가}}{\text{한달 전 주가}} = \text{변화폭 계산 가능}$$

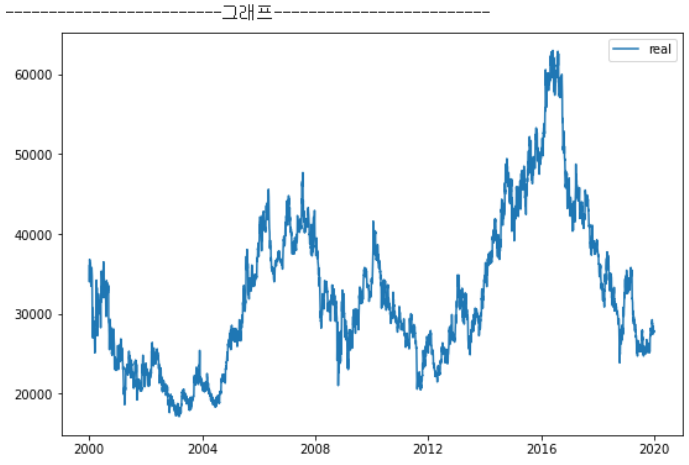
데이터의 Min과 Max가 큰 데이터의 경우 정규화 된 데이터의 차이가

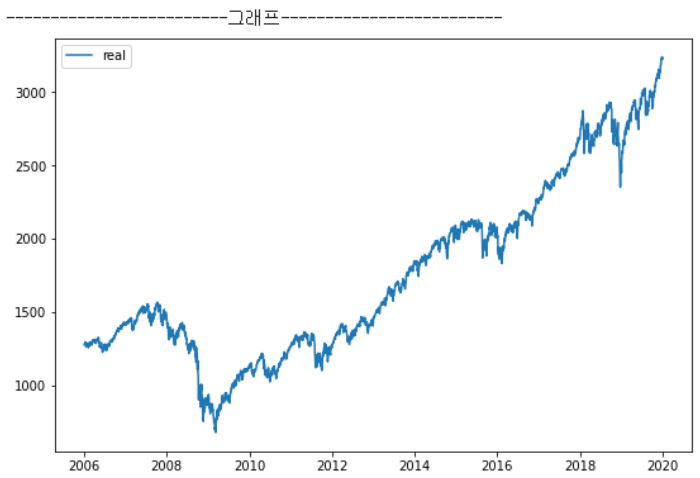
실제 데이터와 다르기 때문 -> 시각화를 해보면 더욱 잘 알 수 있음

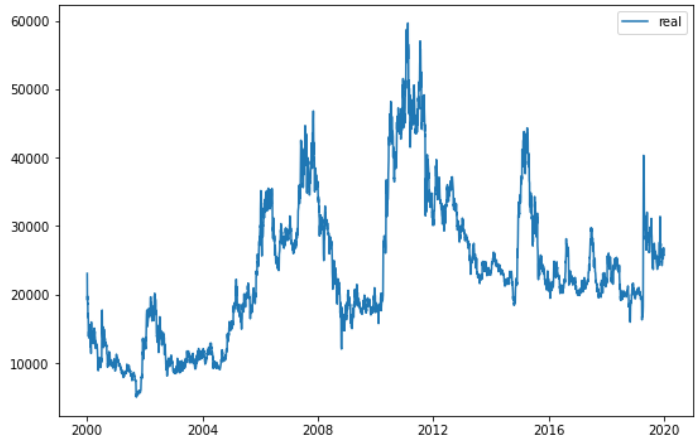
코드 구현

3. 데이터의 유형별 특징을 생각하며 테스트를 해봐야함

(우상향, 우하향, 구간 반복 등.. 의 테스트 별 차이 가장 효율적인 데이터 셋은?)







Data 1 : 한국전력 (주가 data)

Data 2 : S&P 500 (지수 data)

Data 3 : 아시아나 항공 (주가 data)

코드 구현

```
#STOCK_CODE = '015760' # 한국전력
STOCK_CODE = '015760' # S&P 500지수
#STOCK_CODE = 'US500'
```

```
stock = fdr.DataReader(STOCK_CODE, '2000', '2020')
```

```
stock['Month'] = stock.index.month
stock['Year'] = stock.index.year
stock['Day'] = stock.index.day
```

```
print("------------------MinMaxScaler--------------------")
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scale_cols = ['Close']
scaled = scaler.fit_transform(stock[scale_cols])
stock["Close_scaled"] = scaled
stock.tail()
```

```
------------------MinMaxScaler--------------------
```

Date	Open	High	Low	Close	Volume	Change	Month	Year	Day	Close_scaled
2019-12-23	28050	28150	27700	27900	716040	-0.003571	12	2019	23	0.235294
2019-12-24	27850	28000	27650	27900	774460	0.000000	12	2019	24	0.235294
2019-12-26	27850	27850	27450	27650	1262290	-0.008961	12	2019	26	0.229847
2019-12-27	27550	28000	27400	27750	1164446	0.003617	12	2019	27	0.232026
2019-12-30	27850	28000	27600	27800	1188611	0.001802	12	2019	30	0.233115

```
[7]  def zerolistmaker(n): # zeros list
         listofzeros = [0] * n
         return listofzeros

     zeros_list = zerolistmaker(12)
```

```
[8]   array_stock = np.array(stock) # trans numpy
```

```
[9]  def first_month_close(num : int): # # frist_month
       zeros_list = zerolistmaker(12)
       for Open, High, Low, Close, Volume, Change, Month, Year, Day, Close_scaled in array_stock:
        if Year == num:
          for i in range(12):
            if Month == i+1 and zeros_list[i] == 0:
              first_close.append(Close) # Close, Close_scaled
              zeros_list[i] = 1
              break
```

```
[10]  # 한국 전력 2019년 까지 존재

      first_close = []
      first_month_close(2015)
      first_month_close(2016)
      first_month_close(2017)
      first_month_close(2018)
      first_month_close(2019)

      first_close = np.array(first_close)
      print(first_close.shape)
```

```
(60,)
```

코드 구현

```
[23] print("---2015 ~ 2019년 된 월별 금액---")
     print(first_close)
     rate_list = []

     for i in range(len(first_close)):
       # i = index, j = value
       if i == (len(first_close)-1) :
         break
       rate_of_change = (first_close[i] - first_close[i+1]) / first_close[i] * (-1)
       rate_list.append(rate_of_change)
     print("---------변화율 리스트----------")
     rate_list = np.array(rate_list)
     print(rate_list)
```

```
---2015 ~ 2019년 된 월별 금액---
[42700. 43500. 44350. 45950. 47950. 45500. 46350. 52200. 48500. 49000.
 50700. 49750. 50000. 53200. 60000. 59100. 61200. 60400. 59400. 61200.
 57500. 54500. 49400. 44500. 43900. 42450. 43150. 46700. 44750. 44000.
 40700. 44900. 43100. 38100. 38550. 37950. 37750. 36000. 32900. 33250.
 37750. 32900. 32450. 32350. 30300. 29350. 26900. 29750. 34050. 33900.
 35800. 30350. 28050. 26650. 25350. 26650. 25450. 26050. 25350. 28250.]
---------변화율 리스트----------
[ 0.01873536  0.01954023  0.03607666  0.04352557 -0.05109489  0.01868132
  0.12621359 -0.07088123  0.01030928  0.03469388 -0.01873767  0.00502513
  0.064       0.12781955 -0.015       0.03553299 -0.0130719  -0.01655629
  0.03030303 -0.06045752 -0.05217391 -0.09357798 -0.09919028 -0.01348315
 -0.03302961  0.01648999  0.08227115 -0.04175589 -0.01675978 -0.075
  0.1031941  -0.04008909 -0.11600928  0.01181102 -0.0155642  -0.00527009
 -0.04635762 -0.08611111  0.0106383   0.13533835 -0.12847682 -0.01367781
 -0.00308166 -0.0633694  -0.03135314 -0.0834753   0.10594796  0.14453782
 -0.00440529  0.0560472  -0.15223464 -0.07578254 -0.04991087 -0.04878049
  0.05128205 -0.04502814  0.02357564 -0.0268714   0.11439842]
```

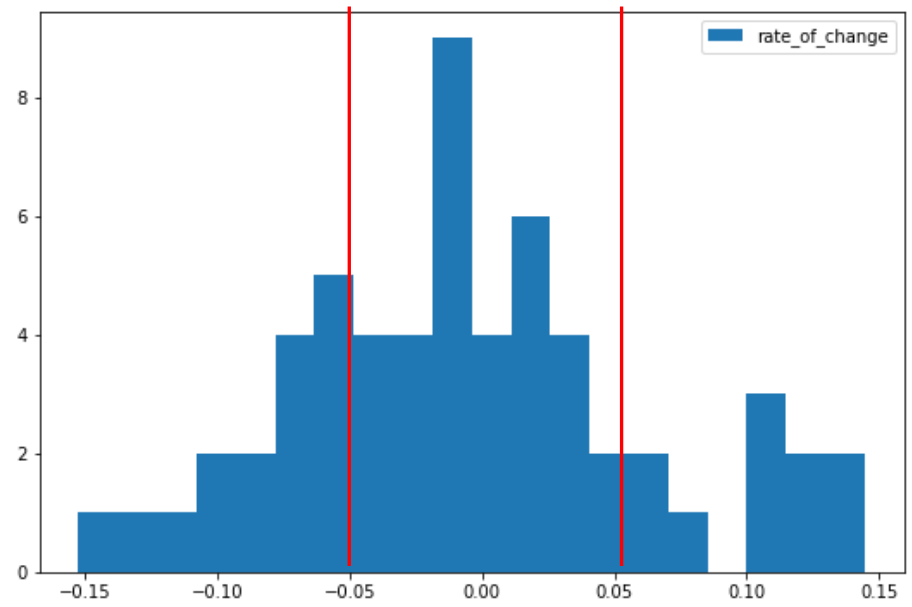
```
[17] print("---2015 ~ 2019년 정규화 된 월별 금액---")
     print(first_close)
     rate_list = []

     for i in range(len(first_close)):
       # i = index, j = value
       if i == (len(first_close)-1) :
         break
       rate_of_change = (first_close[i] - first_close[i+1]) / first_close[i] * (-1)
       rate_list.append(rate_of_change)
     print("---------변화율 리스트----------")
     rate_list = np.array(rate_list)
     print(rate_list)
```

```
---2015 ~ 2019년 정규화 된 월별 금액---
[0.5577342  0.5751634  0.59368192 0.62854031 0.67211329 0.61873638
 0.6372549  0.76470588 0.68409586 0.69498911 0.73202614 0.71132898
 0.7167756  0.78649237 0.93464052 0.91503268 0.96078431 0.94335512
 0.92156863 0.96078431 0.88017429 0.81481481 0.7037037  0.59694989
 0.583878   0.55228758 0.56753813 0.64488017 0.60239651 0.58605664
 0.51416122 0.60566449 0.5664488  0.45751634 0.46732026 0.45424837
 0.44989107 0.41176471 0.34422658 0.35185185 0.44989107 0.34422658
 0.33442266 0.33224401 0.2875817  0.26688453 0.21350763 0.27559913
 0.36928105 0.36601307 0.40740741 0.28867102 0.23856209 0.208061
 0.17973856 0.208061   0.18191721 0.19498911 0.17973856 0.24291939]
---------변화율 리스트----------
[ 0.03125     0.03219697  0.0587156   0.06932409 -0.07941653  0.02992958
  0.2        -0.10541311  0.01592357  0.05329154 -0.02827381  0.00765697
  0.09726444  0.18836565 -0.02097902  0.05       -0.01814059 -0.02309469
  0.04255319 -0.08390023 -0.07425743 -0.13636364 -0.15170279 -0.02189781
 -0.05410448  0.02761341  0.13627639 -0.06587838 -0.02712477 -0.12267658
  0.1779661  -0.0647482  -0.19230769  0.02142857 -0.02797203 -0.00959233
 -0.08474576 -0.16402116  0.0221519   0.27863777 -0.23486683 -0.02848101
 -0.00651466 -0.13442623 -0.0719697  -0.2         0.29081633  0.33992095
 -0.00884956  0.11309524 -0.29144385 -0.17358491 -0.12785388 -0.13612565
  0.15757576 -0.12565445  0.07185629 -0.07821229  0.35151515]
```

코드 구현

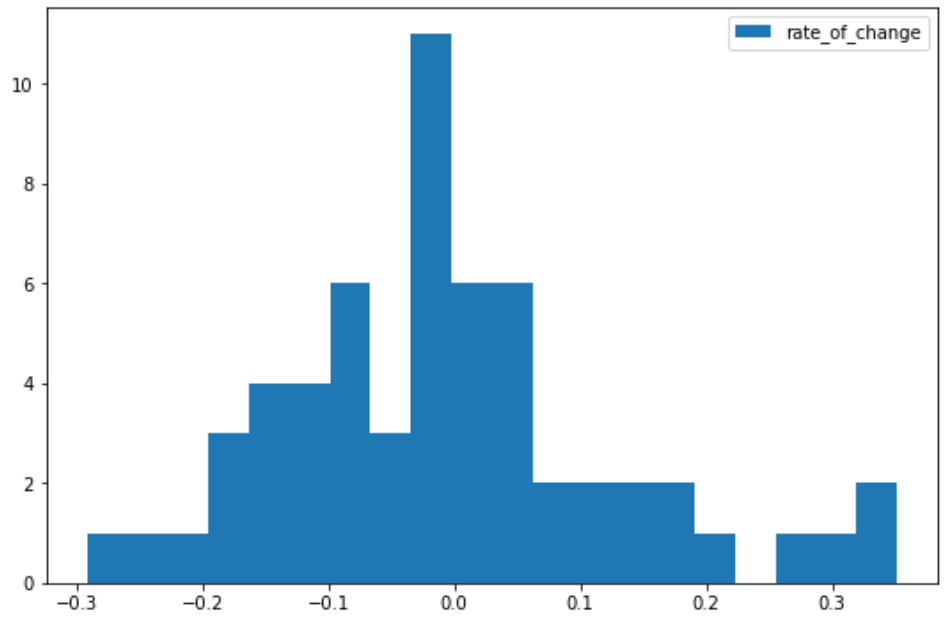
```
[12]  # Close 변화율 시각화
      print("-----------------------실제 변화율 시각화-------------------------")
      plt.figure(figsize=(9, 6))
      plt.hist(rate_list, label = 'rate_of_change',bins = 20)
      plt.legend()
      plt.show()
```

```
findfont: Font family ['NanumGothic'] not found. Falling back to DejaVu Sans.
-----------------------실제 변화율 시각화-------------------------
```




```
[16]  # 변화율 시각화
      print("-----------------------정규화 변화율 시각화-------------------------")
      plt.figure(figsize=(9, 6))
      plt.hist(rate_list, label = 'rate_of_change',bins = 20)
      plt.legend()
      plt.show()
```

```
-----------------------정규화 변화율 시각화-------------------------
```



코드 구현

```
[36] from sklearn.model_selection import train_test_split
     train, test = train_test_split(stock['Close'], test_size=0.3, random_state=0, shuffle=False)
     valid, test = train_test_split(test, test_size=0.66, random_state=0, shuffle=False)
```

```
[37] print("----------train, test 분리 후 shape 출력----------")
     print("train : ", train.shape,"validation :", valid.shape, "test : ", test.shape)

     ----------train, test 분리 후 shape 출력----------
     train :  (3455,) validation : (503,) test :  (978,)
```

```
[101] def four_class (w : list, window_size): # 기준점은 0.05로 잡음
        if (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1) <  -0.05:
          flag = [1,0,0,0]
        elif (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1) > -0.05 and (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1) < 0:
          flag = [0,1,0,0]
        elif (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1) > 0 and (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1) < 0.05:
          flag = [0,0,1,0]
        else :
          flag = [0,0,0,1]
        return flag
```

multi classification

```
[102] def two_class (w : list, window_size): # binary
        test_num = (w[WINDOW_SIZE-1] - w[-1]) / w[WINDOW_SIZE-1] * (-1)
        if test_num > 0:
          flag = [1,0]
        else :
          flag = [0,1]
        return flag
```

Binary classification

```
[29] def min_max_scaler (w : list, window_size):
       a = []
       min, max = find_max_min()
       for i in range(WINDOW_SIZE):
         add = (w[i] - min) / (max - min)
         a.append(add)
       return a
```

```
[30] def find_max_min(): # # frist_month
       min = array_stock[0][3]
       max = array_stock[0][3]
       for Open, High, Low, Close, Volume, Change, Month, Year, Day, Close_scaled in array_stock:
         if Close < min:
           min = Close
         if Close > max:
           max = Close
       return min, max
```

입력 데이터 : 정규화 된 데이터
결과 데이터 : 정규화 전 상승률을 표현

```
[31] day = 1
     week = 5
     month = 20
     asp = month
     def windowed_dataset(series, window_size, batch_size, shuffle):
         series = tf.expand_dims(series, axis=-1)
         ds = tf.data.Dataset.from_tensor_slices(series)
         ds = ds.window(window_size + asp, shift=1, drop_remainder=True)
         ds = ds.flat_map(lambda w: w.batch(window_size + asp))
         if shuffle:
             ds = ds.shuffle(1000)
         ds = ds.map(lambda w: (min_max_scaler(w,WINDOW_SIZE), two_class(w,WINDOW_SIZE))) # two, four
         return ds.batch(batch_size).prefetch(1)
```

코드 구현

```
[34]  import tensorflow as tf

      WINDOW_SIZE= 40
      BATCH_SIZE= 100 # 데이터는 총 50개씩 랜덤으로 뽑음

      train_data = windowed_dataset(train, WINDOW_SIZE, BATCH_SIZE, False)
      valid_data = windowed_dataset(valid, WINDOW_SIZE, BATCH_SIZE, False)
      test_data = windowed_dataset(test, WINDOW_SIZE, BATCH_SIZE, False)
```

```
[35]  print("----------------- y value trans test -----------------")

      for data in train_data.take(1): # month 데이터로 테스트 구분 # 확인 완료
          print(f'데이터셋(X) 구성(batch_size, window_size, feature갯수): {data[0].shape}')
          print(f'데이터셋(Y) 구성(batch_size, window_size, feature갯수): {data[1].shape}')
      print("----------------- code test -----------------")
      for data in train_data.take(1): # month 데이터로 테스트 구분 # 확인 완료
          print(data[0][0][-1])
          print(data[1][0])
      print("----------------- real value test -----------------")
      print(train[39])
      print(train[59])
```

```
----------------- y value trans test -----------------
데이터셋(X) 구성(batch_size, window_size, feature갯수): (100, 40, 1)
데이터셋(Y) 구성(batch_size, window_size, feature갯수): (100, 2)
----------------- code test -----------------
tf.Tensor([0.24400871], shape=(1,), dtype=float64)
tf.Tensor([1 0], shape=(2,), dtype=int32)
----------------- real value test -----------------
28300
30700
```

Test : binary 한달 뒤 상승
-> [1, 0]

```
[43]  import tensorflow as tf

      WINDOW_SIZE= 40
      BATCH_SIZE= 100 # 데이터는 총 50개씩 랜덤으로 뽑음

      train_data = windowed_dataset(train, WINDOW_SIZE, BATCH_SIZE, False)
      valid_data = windowed_dataset(valid, WINDOW_SIZE, BATCH_SIZE, False)
      test_data = windowed_dataset(test, WINDOW_SIZE, BATCH_SIZE, False)
```

```
[44]  print("----------------- y value trans test -----------------")

      for data in train_data.take(1): # month 데이터로 테스트 구분 # 확인 완료
          print(f'데이터셋(X) 구성(batch_size, window_size, feature갯수): {data[0].shape}')
          print(f'데이터셋(Y) 구성(batch_size, window_size, feature갯수): {data[1].shape}')
      print("----------------- code test -----------------")
      for data in train_data.take(1): # month 데이터로 테스트 구분 # 확인 완료
          print(data[0][0][-1])
          print(data[1][0])
      print("----------------- real value test -----------------")
      print(train[39])
      print(train[59])
```

```
----------------- y value trans test -----------------
데이터셋(X) 구성(batch_size, window_size, feature갯수): (100, 40, 1)
데이터셋(Y) 구성(batch_size, window_size, feature갯수): (100, 4)
----------------- code test -----------------
tf.Tensor([0.24400871], shape=(1,), dtype=float64)
tf.Tensor([0 0 0 1], shape=(4,), dtype=int32)
----------------- real value test -----------------
28300
30700
```

Test : multi 5% 이상 상승
-> [0 0 0 1]

주요 논점 (4,5,6)

4. 테스트 한 데이터의 전체 정확도 뿐만 아닌, 각 선택 별 정확도를 알아야 함

5. 평가 데이터의 학습된 모습

6. CNN을 사용한 LSTM Model과 단순 LSTM Model의 학습 차이

LSTM의 unit 개수 및 CNN의 필터 개수에 따른 정확도 차이 테스트

주요 논점(4,5,6)은 실험을 통한 증명을 할 예정입니다.

모델 평가 환경

실험 환경 설정

Data : Month data and Week data

week data (X) : x+40 일 간의 데이터
week data (Y) : x+40일 데이터의 5일 뒤 상승폭 하락폭 학습

month data (X) : x+40 일 간의 데이터
month data (Y) : x+40일 데이터의 20일 뒤 상승폭 하락폭 학습

Y data : Binary classification, 4 – classification

Loss function :

Binary classification : binary cross entropy

4 – classification : categorical cross entropy

Optimaizer : Adam

TEST Model 종류

1. Simple LSTM (unit = 30)
2. CNN(filter = 1) + CNN(filter = 1) + LSTM(unit = 40)
3. CNN(filter = 16) + CNN(filter = 1) + LSTM(unit = 40)
4. CNN(filter = 32) + CNN(filter = 1) + LSTM(unit = 40)
5. CNN(filter = 36) + CNN(filter = 1) + LSTM(unit = 40)
6. CNN(filter = 40) + CNN(filter = 1) + LSTM(unit = 40)
7. CNN(filter = 16) + CNN(filter = 32) + LSTM(unit = 40)
8. CNN(filter = 32) + CNN(filter = 16) + LSTM(unit = 40)

Window data set = 40
Batch Size = 100
Callback : validation loss가 가장 적은 지점

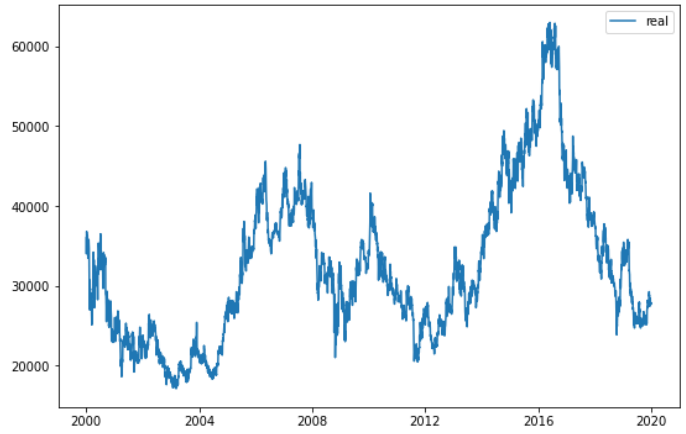
코드 구현

전체 DATA : 2000년 ~ 2020년

Train data : 70%
Validation data : 10%
Test data : 20%

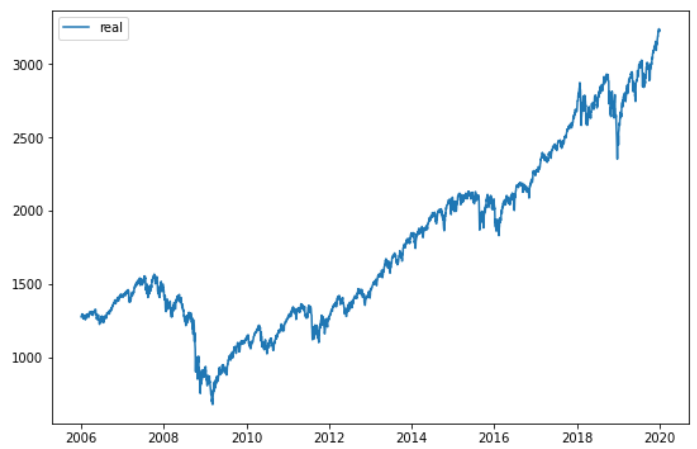
Google Colab에서 Test 진행

----------그래프----------

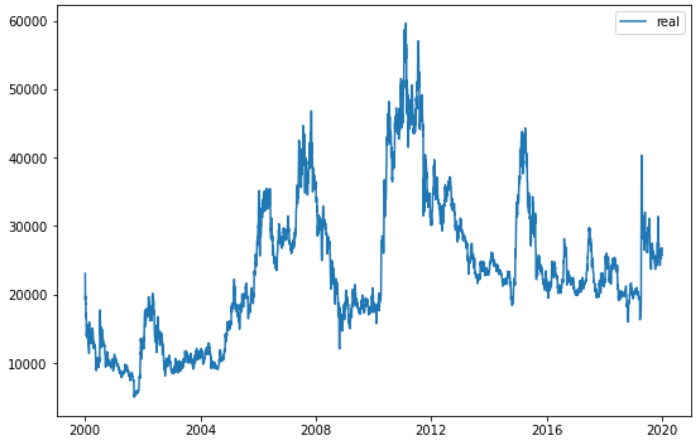


Data 1 : 한국전력 (주가 data)

----------그래프----------



Data 2 : S&P 500 (지수 data)

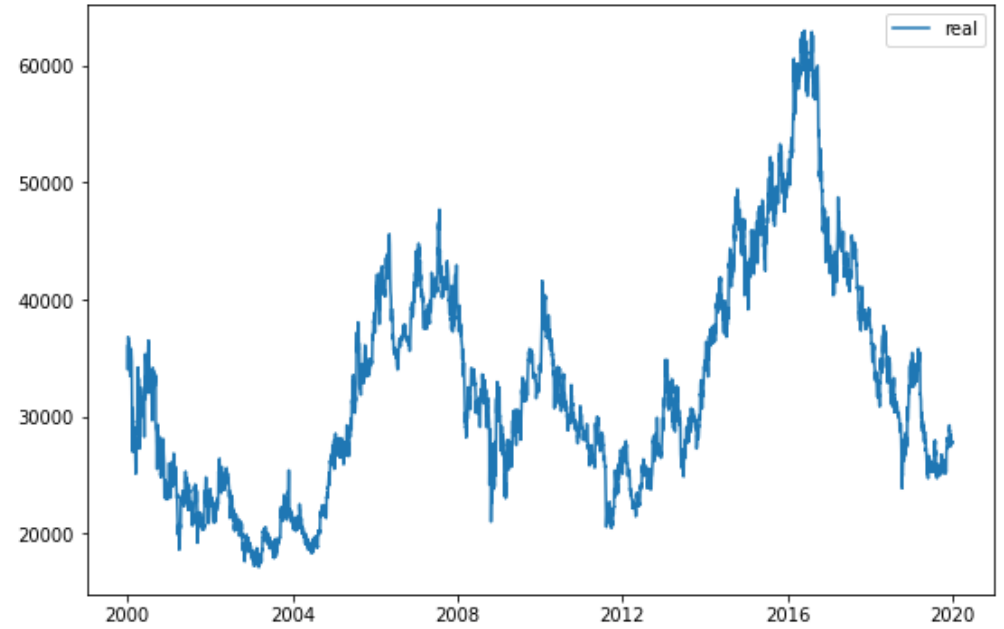


Data 3 : 아시아나 항공 (주가 data)

모델 평가

한국 전력 주가 시각화

----------그래프----------



Data 1 : 한국전력 (주가 data)

코드 구현

단순 LSTM Model의 binary classification 정확도 비교

```
[49] import tensorflow as tf
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

     model = tf.keras.Sequential([
         #tf.keras.layers.Conv1D(filters = 32, kernel_size= 5, padding="same",input_shape=[WINDOW_SIZE, 1]),
         #tf.keras.layers.MaxPool1D(pool_size = 5, padding = "valid"),
         #tf.keras.layers.Conv1D(filters = 25, kernel_size= 4, padding="same"),
         #tf.keras.layers.MaxPool1D(pool_size = 4, padding = "valid"),
         #tf.keras.layers.LSTM(40, activation='tanh', return_sequences= False),
         tf.keras.layers.LSTM(30, activation='tanh', return_sequences= False, input_shape=[WINDOW_SIZE, 1]),
         tf.keras.layers.Dense(10, activation = "relu"),
         tf.keras.layers.Dense(2, activation='softmax'),
     ])
```

```
[50] model.summary()

     Model: "sequential_2"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     lstm_2 (LSTM)                (None, 30)                3840
     _________________________________________________________________
     dense_4 (Dense)              (None, 10)                310
     _________________________________________________________________
     dense_5 (Dense)              (None, 2)                 22
     =================================================================
     Total params: 4,172
     Trainable params: 4,172
     Non-trainable params: 0
     _________________________________________________________________
```

```
[51] # earlystopping은 10번 epoch통안 val_loss 개선이 없다면 학습을 멈춥니다.
     earlystopping = EarlyStopping(monitor='val_loss', patience= 100)
     # val_loss 기준 체크포인터도 생성합니다.
     filename = os.path.join('tmp', 'ckeckpointer.ckpt')
     checkpoint = ModelCheckpoint(filename,  # 가장 val_loss가 적게 떨어졌을 때의 모델 체크
                                  save_weights_only=True,
                                  save_best_only=True,
                                  monitor='val_loss',
                                  verbose=1)
```

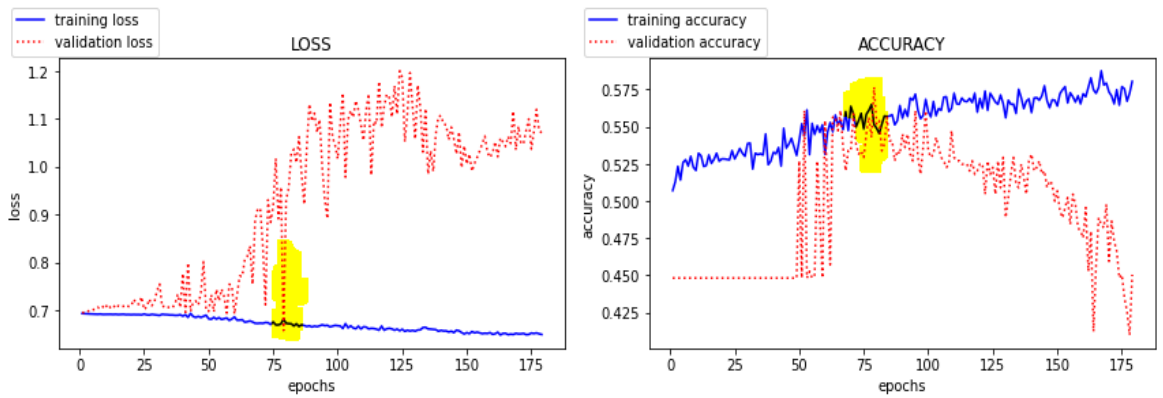
```
[52] optimizer = Adam(0.0005)
     model.compile(loss='binary_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

```
[76] history = model.fit(train_data, validation_data=(valid_data), epochs= 200, callbacks=[checkpoint, earlystopping])

Epoch 00076: val_loss did not improve from 0.69171
Epoch 77/200
34/34 [==============================] - 2s 43ms/step - loss: 0.6691 - accuracy: 0.5610 - val_loss: 0.8913 - val_accuracy: 0.5563

Epoch 00077: val_loss did not improve from 0.69171
Epoch 78/200
34/34 [==============================] - 2s 43ms/step - loss: 0.6710 - accuracy: 0.5648 - val_loss: 0.9576 - val_accuracy: 0.5428

Epoch 00078: val_loss did not improve from 0.69171
Epoch 79/200
34/34 [==============================] - 2s 42ms/step - loss: 0.6814 - accuracy: 0.5512 - val_loss: 0.6575 - val_accuracy: 0.5766

Epoch 00079: val_loss improved from 0.69171 to 0.65748, saving model to tmp/ckeckpointer.ckpt
Epoch 80/200
34/34 [==============================] - 2s 42ms/step - loss: 0.6725 - accuracy: 0.5489 - val_loss: 0.9307 - val_accuracy: 0.5518

Epoch 00080: val_loss did not improve from 0.65748
Epoch 81/200
34/34 [==============================] - 2s 41ms/step - loss: 0.6718 - accuracy: 0.5453 - val_loss: 0.9544 - val_accuracy: 0.5541

Epoch 00081: val_loss did not improve from 0.65748
Epoch 82/200
34/34 [==============================] - 2s 43ms/step - loss: 0.6714 - accuracy: 0.5515 - val_loss: 1.0283 - val_accuracy: 0.5338
```

코드 구현

```
[65]  def vis(history,name) :
          plt.title(f"{name.upper()}")
          plt.xlabel('epochs')
          plt.ylabel(f"{name.lower()}")
          value = history.history.get(name)
          val_value = history.history.get(f"val_{name}",None)
          epochs = range(1, len(value)+1)
          plt.plot(epochs, value, 'b-', label=f'training {name}')
          if val_value is not None :
              plt.plot(epochs, val_value, 'r:', label=f'validation {name}')
          plt.legend(loc='upper center', bbox_to_anchor=(0.05, 1.2) , fontsize=10 , ncol=1)

      def plot_history(history) :
          key_value = list(set([i.split("val_")[-1] for i in list(history.history.keys())]))
          plt.figure(figsize=(12, 4))
          for idx , key in enumerate(key_value) :
              plt.subplot(1, len(key_value), idx+1)
              vis(history, key)
          plt.tight_layout()
          plt.show()
```

```
[77]  plot_history(history)
```




```
[78]  print("---------------- test evaluate ----------------")

      model.evaluate(test_data)


      print("---------------- test label 데이터 뽑기 ----------------")

      label = []

      for data in test_data: # month 데이터로 테스트 구분 # 확인 완료
          for i in data[1]:
              label.append(i)
      label = np.array(label)
      label.shape
```

```
---------------- test evaluate ----------------
10/10 [==============================] - 1s 31ms/step - loss: 0.7412 - accuracy: 0.5462
---------------- test label 데이터 뽑기 ----------------
(919, 2)
```

코드 구현

```
[79]  scale = model.predict(test_data)
      data_columns = ["상승","하락"]
      print(scale.shape)
      evaluate_df = pd.DataFrame(data = scale, columns = data_columns)
      data_columns = ["상승정답", "하락정답"]
      evaluate_df[data_columns] = label
      evaluate_np = np.array(evaluate_df)
```

```
(919, 2)
```

```
[80]  # 2가지 각각의 맞춘 정도를 출력

      # 강한 상승, 약한 상승, 약한 하락, 강한 하락 중 가장 나올 확률이 높은 수

      Down_count = 0
      Up_count = 0

      test_count1 = 0
      test_count2 = 0


      for Up, Down, Up_label, Down_label in evaluate_np:
        max_num = max(Up, Down)
        if max_num == Up:
          Up_count+=1
        elif max_num == Down:
          Down_count+=1
        if Up_label == 1:
            test_count1+=1
        elif Down_label == 1:
            test_count2+=1

      print("----------------------각 상황의 카운터----------------------------")
      print("Up_count : ",Up_count )
      print("Up_label_count = ", test_count1)
      print("Down_count = ", Down_count )
      print("Down_label_count =  ", test_count2 )

      Down_count1 = 0
      Up_count1 = 0
      test_count3 = 0
      test_count4 = 0
```

테스트 한 데이터의 전체 정확도 뿐만 아닌,
각 선택 별 정확도를 알아야 함

```
for Up, Down, Up_label, Down_label in evaluate_np:
  max_num = max(Up, Down)
  if max_num == Up:
    Up_count1+=1
    if Up_label == 1:
      test_count3+=1
  elif max_num == Down:
    Down_count1+=1
    if Down_label == 1:
      test_count4+=1

print("----------------------각 상황의 예측 정확도----------------------------")
print("Up evaluate rate = ", test_count3/Up_count1)
if Down_count != 0:
  print("Down evaluate rate = ", test_count4/Down_count1)
```

```
----------------------각 상황의 카운터----------------------------
Up_count :  402
Up_label_count =  351
Down_count =  517
Down_label_count =  568
----------------------각 상황의 예측 정확도----------------------------
Up evaluate rate =  0.417910447761194
Down evaluate rate =  0.6460348162475822
```

코드 구현

2. 단순 LSTM Model의 4가지 classification 정확도 비교

```
[92] import tensorflow as tf
     from tensorflow.keras.optimizers import Adam
     from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

     model = tf.keras.Sequential([
         #tf.keras.layers.Conv1D(filters = 32, kernel_size= 5, padding="same",input_shape=[WINDOW_SIZE, 1]),
         #tf.keras.layers.MaxPool1D(pool_size = 5, padding = "valid"),
         #tf.keras.layers.Conv1D(filters = 25, kernel_size= 4, padding="same"),
         #tf.keras.layers.MaxPool1D(pool_size = 4, padding = "valid"),
         #tf.keras.layers.LSTM(40, activation='tanh', return_sequences= False),
         tf.keras.layers.LSTM(30, activation='tanh', return_sequences= False, input_shape=[WINDOW_SIZE, 1]),
         tf.keras.layers.Dense(10, activation = "relu"),
         tf.keras.layers.Dense(4, activation='softmax'),
     ])
```

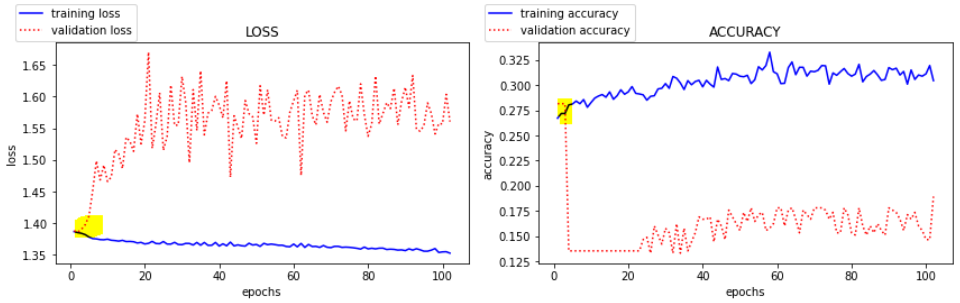
```
[93] model.summary()

     Model: "sequential_6"
     _________________________________________________________________
     Layer (type)                 Output Shape              Param #
     =================================================================
     lstm_6 (LSTM)                (None, 30)                3840
     _________________________________________________________________
     dense_12 (Dense)             (None, 10)                310
     _________________________________________________________________
     dense_13 (Dense)             (None, 4)                 44
     =================================================================
     Total params: 4,194
     Trainable params: 4,194
     Non-trainable params: 0
     _________________________________________________________________
```

```
[161] optimizer = Adam(0.0005) # 0.0001, 0.0005
      model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

```
[162] history = model.fit(train_data, validation_data=(valid_data), epochs= 200, callbacks=[checkpoint, earlystopping])

      Epoch 1/200
      34/34 [==============================] - 3s 58ms/step - loss: 1.3866 - accuracy: 0.2671 - val_loss: 1.3878 - val_accuracy: 0.2815

      Epoch 00001: val_loss improved from inf to 1.38783, saving model to tmp/ckeckpointer.ckpt
      Epoch 2/200
      34/34 [==============================] - 2s 49ms/step - loss: 1.3850 - accuracy: 0.2718 - val_loss: 1.3859 - val_accuracy: 0.2815

      Epoch 00002: val_loss improved from 1.38783 to 1.38588, saving model to tmp/ckeckpointer.ckpt
      Epoch 3/200
      34/34 [==============================] - 2s 49ms/step - loss: 1.3839 - accuracy: 0.2718 - val_loss: 1.3902 - val_accuracy: 0.2815

      Epoch 00003: val_loss did not improve from 1.38588
      Epoch 4/200
      34/34 [==============================] - 2s 49ms/step - loss: 1.3818 - accuracy: 0.2803 - val_loss: 1.3975 - val_accuracy: 0.1351

      Epoch 00004: val_loss did not improve from 1.38588
      Epoch 5/200
      34/34 [==============================] - 2s 50ms/step - loss: 1.3781 - accuracy: 0.2812 - val_loss: 1.4109 - val_accuracy: 0.1351

      Epoch 00005: val_loss did not improve from 1.38588
```

코드 구현

```
[163] plot_history(history)
```




```
[164] print("---------------- test evaluate ----------------")

      model.evaluate(test_data)


      print("---------------- test label 데이터 뽑기 ----------------")

      label = []

      for data in test_data: # month 데이터로 테스트 구분 # 확인 완료
          for i in data[1]:
            label.append(i)
      label = np.array(label)
      label.shape
```

```
---------------- test evaluate ----------------
10/10 [==============================] - 1s 37ms/step - loss: 1.3642 - accuracy: 0.2372
---------------- test label 데이터 뽑기 ----------------
(919, 4)
```

```
[98]  scale = model.predict(test_data)
      data_columns = ["강한 하락","약한 하락","약한 상승","강한 상승"]
      print(scale.shape)
      evaluate_df = pd.DataFrame(data = scale, columns = data_columns)
      data_columns = ["강한 하락정답","약한 하락정답","약한 상승정답","강한 상승정답"]
      evaluate_df[data_columns] = label
```

```
(919, 4)
```

```
print("High_Down Count = ", High_Down_count)
print("High_Down_label = ",  test_count3)
print("Low_Down Count = ", Low_Down_count)
print("Low_Down_label = ", test_count2)
print("High_Up Count = ", High_up_count)
print("High_Up_label = ", test_count4)
print("Low_Up Count = ", Low_up_count)
print("Low_Up_label = ",  test_count1)
```

```
High_Down Count =  411
High_Down_label =  272
Low_Down Count =  7
Low_Down_label =  289
High_Up Count =  310
High_Up_label =  148
Low_Up Count =  191
Low_Up_label =  210
-------------------------각 상황의 예측 정확도-----------------------------
Low_Up evaluate rate =  0.17277486910994763
Low_Down evaluate rate =  0.14285714285714285
High_Down evaluate rate = 0 0.26034063260340634
High_Up evaluate rate = 0  0.24838709677419354
```

코드 구현

2. CNN을 포함한 LSTM Model의 binary classification 정확도 비교

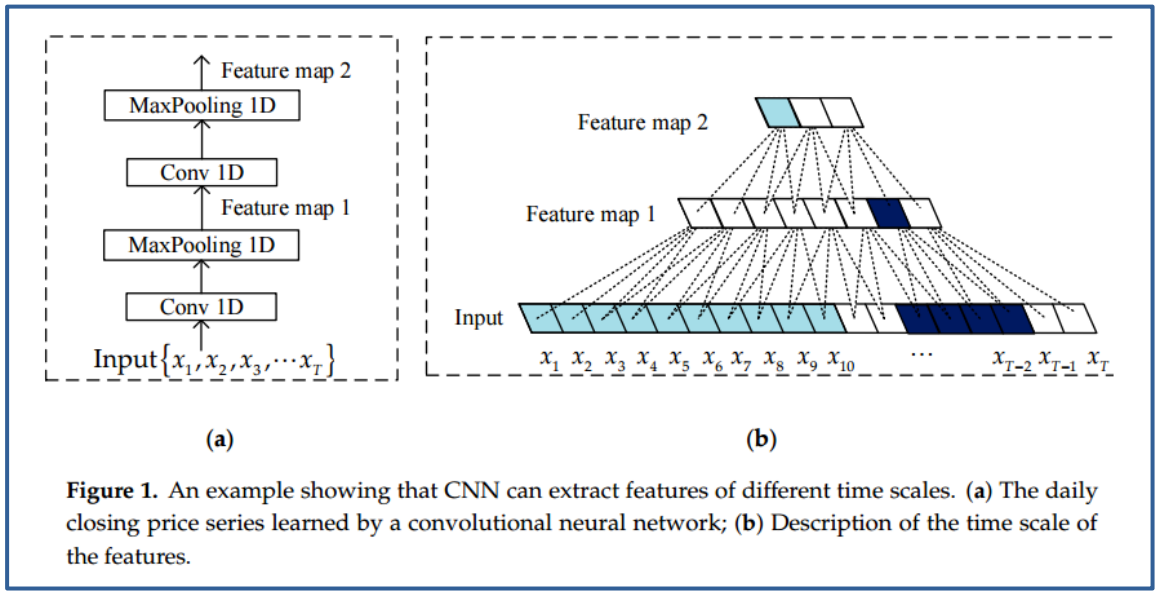


Figure 1. An example showing that CNN can extract features of different time scales. **(a)** The daily closing price series learned by a convolutional neural network; **(b)** Description of the time scale of the features.

```
[191] import tensorflow as tf
      from tensorflow.keras.optimizers import Adam
      from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

      model = tf.keras.Sequential([
          tf.keras.layers.Conv1D(filters = 1, kernel_size= 5, padding="same",input_shape=[WINDOW_SIZE, 1]),
          tf.keras.layers.MaxPool1D(pool_size = 5, padding = "valid"),
          tf.keras.layers.Conv1D(filters = 1, kernel_size= 4, padding="same"),
          tf.keras.layers.MaxPool1D(pool_size = 4, padding = "valid"),
          tf.keras.layers.LSTM(40, activation='tanh', return_sequences= False),
          #tf.keras.layers.LSTM(30, activation='tanh', return_sequences= False, input_shape=[WINDOW_SIZE, 1]),
          tf.keras.layers.Dense(10, activation = "relu"),
          tf.keras.layers.Dense(2, activation='softmax'),
      ])
```

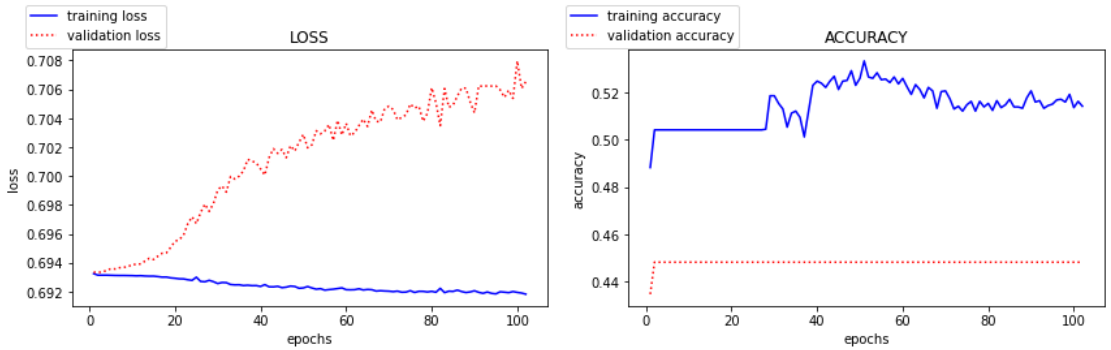
```
[192] model.summary()

      Model: "sequential_17"
      _________________________________________________________________
      Layer (type)                 Output Shape              Param #
      =================================================================
      conv1d_14 (Conv1D)           (None, 40, 1)             6
      _________________________________________________________________
      max_pooling1d_14 (MaxPooling (None, 8, 1)              0
      _________________________________________________________________
      conv1d_15 (Conv1D)           (None, 8, 1)              5
      _________________________________________________________________
      max_pooling1d_15 (MaxPooling (None, 2, 1)              0
      _________________________________________________________________
      lstm_19 (LSTM)               (None, 40)                6720
      _________________________________________________________________
      dense_34 (Dense)             (None, 10)                410
      _________________________________________________________________
      dense_35 (Dense)             (None, 2)                 22
      =================================================================
      Total params: 7,163
      Trainable params: 7,163
      Non-trainable params: 0
      _________________________________________________________________
```

코드 구현

```
[195] history = model.fit(train_data, validation_data=(valid_data), epochs= 200, callbacks=[checkpoint, earlystopping])
```

```
Epoch 1/200
34/34 [==============================] - 4s 47ms/step - loss: 0.6932 - accuracy: 0.4882 - val_loss: 0.6933 - val_accuracy: 0.4347

Epoch 00001: val_loss improved from inf to 0.69335, saving model to tmp/ckeckpointer.ckpt
Epoch 2/200
34/34 [==============================] - 1s 38ms/step - loss: 0.6931 - accuracy: 0.5041 - val_loss: 0.6933 - val_accuracy: 0.4482

Epoch 00002: val_loss improved from 0.69335 to 0.69333, saving model to tmp/ckeckpointer.ckpt
Epoch 3/200
34/34 [==============================] - 1s 38ms/step - loss: 0.6931 - accuracy: 0.5041 - val_loss: 0.6934 - val_accuracy: 0.4482

Epoch 00003: val_loss did not improve from 0.69333
Epoch 4/200
34/34 [==============================] - 1s 40ms/step - loss: 0.6931 - accuracy: 0.5041 - val_loss: 0.6935 - val_accuracy: 0.4482
```

```
plot_history(history)
```




```
[200] print("---------------- test evaluate ----------------")

      model.evaluate(test_data)


      print("---------------- test label 데이터 뽑기 ----------------")

      label = []

      for data in test_data: # month 데이터로 테스트 구분 # 확인 완료
          for i in data[1]:
              label.append(i)
      label = np.array(label)
      label.shape
```

```
---------------- test evaluate ----------------
10/10 [==============================] - 1s 80ms/step - loss: 0.6690 - accuracy: 0.6551
---------------- test label 데이터 뽑기 ----------------
(919, 2)
```

```
----------------------각 상황의 카운터----------------------------
Up_count :  138
Up_label_count =  351
Down_count =  781
Down_label_count =  568
----------------------각 상황의 예측 정확도----------------------------
Up evaluate rate =  0.6231884057971014
Down evaluate rate =  0.6606914212548015
```

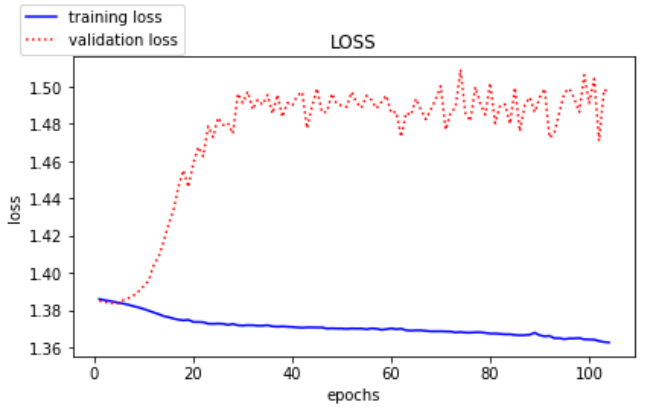
코드 구현

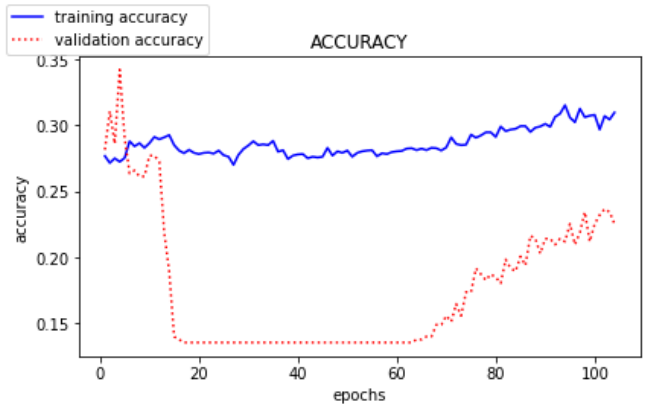
```
[206] import tensorflow as tf
      from tensorflow.keras.optimizers import Adam
      from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

      model = tf.keras.Sequential([
          tf.keras.layers.Conv1D(filters = 1, kernel_size= 5, padding="same",input_shape=[WINDOW_SIZE, 1]),
          tf.keras.layers.MaxPool1D(pool_size = 5, padding = "valid"),
          tf.keras.layers.Conv1D(filters = 1, kernel_size= 4, padding="same"),
          tf.keras.layers.MaxPool1D(pool_size = 4, padding = "valid"),
          tf.keras.layers.LSTM(40, activation='tanh', return_sequences= False),
          #tf.keras.layers.LSTM(30, activation='tanh', return_sequences= False, input_shape=[WINDOW_SIZE, 1]),
          tf.keras.layers.Dense(10, activation = "relu"),
          tf.keras.layers.Dense(4, activation='softmax'),
      ])
```

```
[207] model.summary()

      Model: "sequential_18"
      _________________________________________________________________
      Layer (type)                 Output Shape              Param #
      =================================================================
      conv1d_16 (Conv1D)           (None, 40, 1)             6
      _________________________________________________________________
      max_pooling1d_16 (MaxPooling (None, 8, 1)              0
      _________________________________________________________________
      conv1d_17 (Conv1D)           (None, 8, 1)              5
      _________________________________________________________________
      max_pooling1d_17 (MaxPooling (None, 2, 1)              0
      _________________________________________________________________
      lstm_20 (LSTM)               (None, 40)                6720
      _________________________________________________________________
      dense_36 (Dense)             (None, 10)                410
      _________________________________________________________________
      dense_37 (Dense)             (None, 4)                 44
      =================================================================
      Total params: 7,185
      Trainable params: 7,185
      Non-trainable params: 0
      _________________________________________________________________
```

```
[209] optimizer = Adam(0.0005) # 0.0001, 0.0005
      model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
```

```
[210] history = model.fit(train_data, validation_data=(valid_data), epochs= 200, callbacks=[checkpoint, earlystopping])

      Epoch 1/200
      34/34 [==============================] - 3s 55ms/step - loss: 1.3860 - accuracy: 0.2768 - val_loss: 1.3851 - val_accuracy: 0.2815

      Epoch 00001: val_loss improved from inf to 1.38506, saving model to tmp/ckeckpointer.ckpt
      Epoch 2/200
      34/34 [==============================] - 2s 47ms/step - loss: 1.3854 - accuracy: 0.2715 - val_loss: 1.3842 - val_accuracy: 0.3108

      Epoch 00002: val_loss improved from 1.38506 to 1.38420, saving model to tmp/ckeckpointer.ckpt
      Epoch 3/200
      34/34 [==============================] - 2s 45ms/step - loss: 1.3849 - accuracy: 0.2750 - val_loss: 1.3843 - val_accuracy: 0.2860

      Epoch 00003: val_loss did not improve from 1.38420
      Epoch 4/200
      34/34 [==============================] - 2s 47ms/step - loss: 1.3845 - accuracy: 0.2724 - val_loss: 1.3834 - val_accuracy: 0.3423

      Epoch 00004: val_loss improved from 1.38420 to 1.38338, saving model to tmp/ckeckpointer.ckpt
      Epoch 5/200
      34/34 [==============================] - 2s 46ms/step - loss: 1.3839 - accuracy: 0.2753 - val_loss: 1.3838 - val_accuracy: 0.2905

      Epoch 00005: val_loss did not improve from 1.38338
      Epoch 6/200
      34/34 [==============================] - 2s 47ms/step - loss: 1.3835 - accuracy: 0.2880 - val_loss: 1.3857 - val_accuracy: 0.2635
```





코드 구현

```
[212] print("----------------- test evaluate -----------------")

      model.evaluate(test_data)


      print("----------------- test label 데이터 뽑기 -----------------")

      label = []

      for data in test_data: # month 데이터로 테스트 구분 # 확인 완료
          for i in data[1]:
            label.append(i)
      label = np.array(label)
      label.shape
```

```
----------------- test evaluate -----------------
10/10 [==============================] - 1s 34ms/step - loss: 1.3775 - accuracy: 0.3341
----------------- test label 데이터 뽑기 -----------------
(919, 4)
```

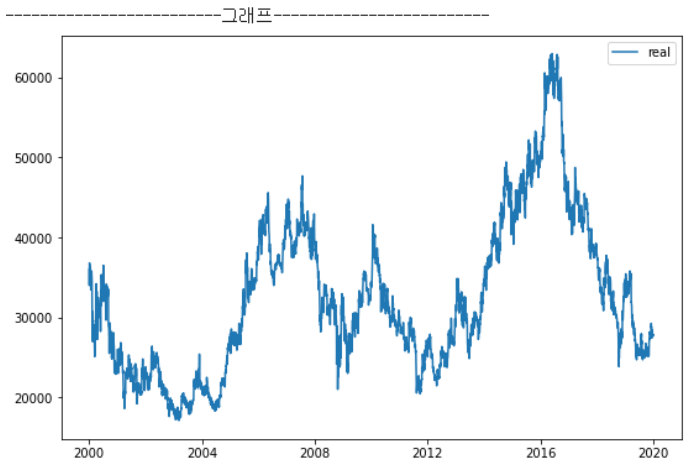
```
print("High_Down Count = ", High_Down_count)
print("High_Down_label = ",  test_count3)
print("Low_Down Count = ", Low_Down_count)
print("Low_Down_label = ", test_count2)
print("High_Up Count = ", High_up_count)
print("High_Up_label = ", test_count4)
print("Low_Up Count = ", Low_up_count)
print("Low_Up_label = ",  test_count1)
```

```
High_Down Count =  257
High_Down_label =  272
Low_Down Count =  40
Low_Down_label =  289
High_Up Count =  252
High_Up_label =  148
Low_Up Count =  370
Low_Up_label =  210
```

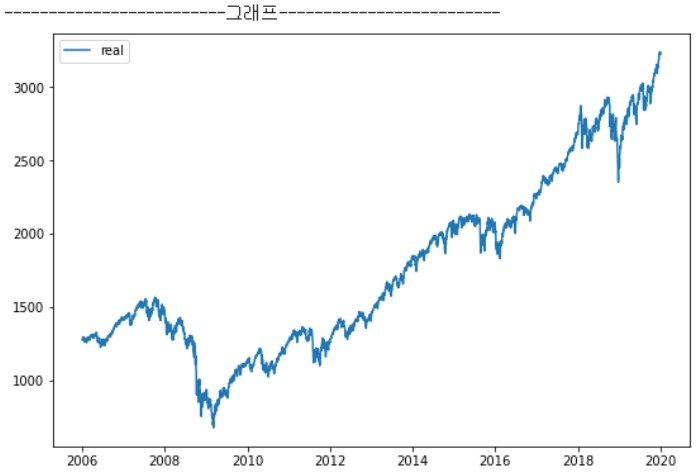
```
------------------------각 상황의 예측 정확도-----------------------------
Low_Up evaluate rate =  0.2810810810810811
Low_Down evaluate rate =  0.225
High_Down evaluate rate = 0 0.4474708171206226
High_Up evaluate rate = 0  0.3134920634920635
```

평가 결과

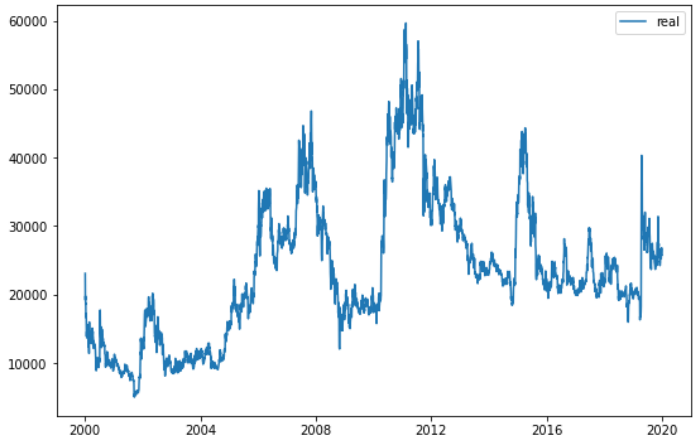
실험 결과



Data 1 : 한국전력 (주가 data)



Data 2 : S&P 500 (지수 data)



Data 3 : 아시아나 항공 (주가 data)

순서 : 한국 전력, S&P 500, 아시아나 항공

1. Month data test
2. Week data test

실험 결과(Month data)

Data 1 : 한국전력 (주가 data) - month

Model - 2th	train	validation	test	상승	하락
LSTM	0.5512	0.5766	0.5462	0.41	0.64
CNN(1,1) + LSTM	0.5041	0.4482	0.6551	0.62	0.66
CNN(16,1) + LSTM	0.5006	0.4482	0.6464	0.63	0.65
CNN(24,1) + LSTM	0.4806	0.5518	0.6529	0.63	0.66
CNN(32,1) + LSTM	0.5041	0.4482	0.6355	0.54	0.66
CNN(36,1) + LSTM	0.5015	0.4482	0.667	0.63	0.67
CNN(40,1) + LSTM	0.4876	0.4482	0.6616	0.62	0.67
CNN(36,18) + LSTM	0.5536	0.5113	0.6039	0.46	0.63
CNN(18,36) + LSTM	0.4988	0.4482	0.5974	0.45	0.63

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.2718	0.2815	0.2372	0.24	0.17	0.14	0.26
CNN(1,1) + LSTM	0.2724	0.3423	0.3341	0.31	0.28	0.22	0.44
CNN(16,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(24,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(32,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(36,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(40,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(36,18) + LSTM	0.2718	0.2815	0.3101	0.25	0.18	0.31	0.34
CNN(18,36) + LSTM	0.4122	0.4167	0.5212	0	0	0.52	0

실험 결과(Month data)

Data 2 : S&P 500 (지수 data)

Model - 2th	train	validation	test	상승	하락
LSTM	0.6291	0.7167	0.7011	0.7	0
CNN(1,1) + LSTM	0.6291	0.7167	0.7011	0.7	0
CNN(16,1) + LSTM	0.662	0.7167	0.3678	0.73	0.3
CNN(24,1) + LSTM	0.6291	0.7167	0.4304	0.68	0.2
CNN(32,1) + LSTM	0.6291	0.7167	0.5164	0.86	0.37
CNN(36,1) + LSTM	0.6291	0.7167	0.6228	0.79	0.41
CNN(40,1) + LSTM	0.6291	0.7167	0.446	0.83	0.34
CNN(36,18) + LSTM	0.6291	0.7167	0.7011	0.7	0
CNN(18,36) + LSTM	0.6291	0.7167	0.446	0.83	0.34

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.5335	0.6533	0.6322	0	0.63	0	0
CNN(1,1) + LSTM	0.5335	0.6533	0.4304	0	0.77	0.28	0
CNN(16,1) + LSTM	0.5335	0.6533	0.4679	0	0.77	0.3	0
CNN(24,1) + LSTM	0.5335	0.6533	0.4304	0	0.68	0.31	0
CNN(32,1) + LSTM	0.5335	0.6533	0.6322	0	0.63	0	0
CNN(36,1) + LSTM	0.5335	0.6533	0.6103	0	0.62	0	0
CNN(40,1) + LSTM	0.5335	0.6553	0.3756	0	0.79	0.26	0
CNN(36,18) + LSTM	0.5335	0.6553	0.6322	0	0.63	0	0
CNN(18,36) + LSTM	0.5335	0.6553	0.6322	0	0.63	0	0

실험 결과(Month data)

Data 3 : 아시아나 항공 (주가 data)

Model - 2th	train	validation	test	상승	하락
LSTM	0.542	0.6059	0.5691	1	0.55
CNN(1,1) + LSTM	0.5502	0.6059	0.5365	0.71	0.53
CNN(16,1) + LSTM	0.5461	0.6059	0.5571	0.87	0.55
CNN(24,1) + LSTM	0.5502	0.6059	0.5702	0.9	0.55
CNN(32,1) + LSTM	0.5488	0.6032	0.5365	0.54	0.54
CNN(36,1) + LSTM	0.562	0.6059	0.5441	0.54	0.54
CNN(40,1) + LSTM	0.5402	0.6059	0.5299	0.44	0.53
CNN(36,18) + LSTM	0.5859	0.7568	0.5386	0.56	0.53
CNN(18,36) + LSTM	0.5541	0.6419	0.5332	0.5	0.66

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.3811	0.509	0.3787	0.33	0	0.42	0
CNN(1,1) + LSTM	0.3959	0.4617	0.3961	0.35	0	0.43	0
CNN(16,1) + LSTM	0.3938	0.4302	0.3874	0.52	0	0.38	0
CNN(24,1) + LSTM	0.3985	0.482	0.4	0.35	0	0.45	0
CNN(32,1) + LSTM	0.3962	0.536	0.3493	0.3	0	0.42	0
CNN(36,1) + LSTM	0.4044	0.5068	0.3351	0.28	0	0.42	0
CNN(40,1) + LSTM	0.3979	0.4932	0.3047	0.28	0	0.37	0
CNN(36,18) + LSTM	0.395	0.4887	0.3025	0.26	0	0.41	0
CNN(18,36) + LSTM	0.4477	0.4865	0.3645	0.21	0	0.39	0

실험 결과 (Week data)

Data 1 : 한국전력 (주가 data) - week

week - 2th	train	validation	test	상승	하락
LSTM	0.5312	0.4706	0.5493	0.37	0.56
CNN(1,1) + LSTM	0.5318	0.4706	0.5685	0	0.57
CNN(16,1) + LSTM	0.5312	0.4706	0.5707	0	0.56
CNN(24,1) + LSTM	0.5204	0.4706	0.5685	0	0.57
CNN(32,1) + LSTM	0.5303	0.4706	0.5685	0	0.57
CNN(36,1) + LSTM	0.5248	0.4706	0.5685	0	0.57
CNN(40,1) + LSTM	0.5312	0.4706	0.5685	0	0.57
CNN(36,18) + LSTM	0.5239	0.4706	0.5685	0	0.57
CNN(18,36) + LSTM	0.5286	0.4706	0.5685	0	0.57

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(1,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(16,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(24,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(32,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(36,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(40,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(36,18) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(18,36) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0

실험 결과 (Week data)

Data 2 : S&P 500 (지수 data)

week - 2th	train	validation	test	상승	하락
LSTM	0.5715	0.6032	0.6468	0.65	0
CNN(1,1) + LSTM	0.5715	0.6032	0.6468	0.65	0
CNN(16,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(24,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(32,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(36,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(40,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(36,18) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(18,36) + LSTM	0.5715	0.6032	0.6488	0.65	0

week - 2th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(1,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(16,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(24,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(32,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(36,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(40,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(36,18) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(18,36) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0

실험 결과 (Week data)

Data 3 : 아시아나 항공 (주가 data)

week - 2th	train	validation	test	상승	하락
LSTM	0.5267	0.5621	0.5332	0.17	0.54
CNN(1,1) + LSTM	0.5372	0.5621	0.5439	0	0.54
CNN(16,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(24,1) + LSTM	0.5334	0.5621	0.5375	0	0.54
CNN(32,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(36,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(40,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(36,18) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(18,36) + LSTM	0.5267	0.5621	0.5375	0	0.54

week - 2th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.4616	0.5185	0.4925	0	0	0.49	0
CNN(1,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(16,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(24,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(32,1) + LSTM	0.4616	0.5185	0.4925	0	0	0.49	0
CNN(36,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(40,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(36,18) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(18,36) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0

실험 결과

제일 학습이 잘 된 모델의 기준점

1. 주가의 상승 및 하락을 예측하는 모델은 완벽하게 교육될 수 없습니다.
(지난 40개의 주가가 예측하려는 주가와 완벽한 인과관계를 맺고 있지 않기 때문)

따라서 Test data set의 정확도 뿐만 아닌, Train data와 validation data의 정확도의 결과를 같이 평가해야합니다.
즉 Test set의 평가가 50%이상 이라도, 좋은 결과가 아닐 수 있다.
(train data와 validation data, test data 전부 50%의 정확도 이상을 가져야 합니다.)

2. 주식 매매는 상승 및 하락 둘 중 한가지 경우만 예측을 잘하더라도 수익을 낼 수 있는 구조이기 때문에,
상승과 하락 중 한가지의 특성이라도 예측율이 높은 경우 좋은 모델이라고 평가 할 수 있습니다.

주의해야할 점 : 예측 개수입니다. 가령 Test 데이터가 500개가 있음에도 불구하고,
10개 미만의 데이터를 예측하여 선택의 정확성을 높이는 경우는 제외시킵니다.

실험 결과

Week data Test Result

평가에서 사용된 3가지의 데이터들로 테스트 한 결과
모델은 대부분 한가지의 결과로 학습되는 것을 확인할 수 있습니다.
이는 여러가지의 이유가 존재하겠지만, 입력데이터와 정답 데이터 간의
상관관계가 형성이 되어있지 않기 때문에,
학습이 어려운 것으로 예측됩니다.

Week data Test의 경우는 따라서 Train이 어려운 것을 확인하였습니다.

한국전력

week - 2th	train	validation	test	상승	하락
LSTM	0.5312	0.4706	0.5493	0.37	0.56
CNN(1,1) + LSTM	0.5318	0.4706	0.5685	0	0.57
CNN(16,1) + LSTM	0.5312	0.4706	0.5707	0	0.56
CNN(24,1) + LSTM	0.5204	0.4706	0.5685	0	0.57
CNN(32,1) + LSTM	0.5303	0.4706	0.5685	0	0.57
CNN(36,1) + LSTM	0.5248	0.4706	0.5685	0	0.57
CNN(40,1) + LSTM	0.5312	0.4706	0.5685	0	0.57
CNN(36,18) + LSTM	0.5239	0.4706	0.5685	0	0.57
CNN(18,36) + LSTM	0.5286	0.4706	0.5685	0	0.57

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(1,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(16,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(24,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(32,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(36,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(40,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(36,18) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0
CNN(18,36) + LSTM	0.4987	0.4488	0.5439	0	0	0.5439	0

S&P 500

week - 2th	train	validation	test	상승	하락
LSTM	0.5715	0.6032	0.6468	0.65	0
CNN(1,1) + LSTM	0.5715	0.6032	0.6468	0.65	0
CNN(16,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(24,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(32,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(36,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(40,1) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(36,18) + LSTM	0.5715	0.6032	0.6488	0.65	0
CNN(18,36) + LSTM	0.5715	0.6032	0.6488	0.65	0

week - 2th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(1,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(16,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(24,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(32,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(36,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(40,1) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(36,18) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0
CNN(18,36) + LSTM	0.5488	0.6032	0.6407	0	0.64	0	0

아시아나

week - 2th	train	validation	test	상승	하락
LSTM	0.5267	0.5621	0.5332	0.17	0.54
CNN(1,1) + LSTM	0.5372	0.5621	0.5439	0	0.54
CNN(16,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(24,1) + LSTM	0.5384	0.5621	0.5375	0	0.54
CNN(32,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(36,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(40,1) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(36,18) + LSTM	0.5267	0.5621	0.5375	0	0.54
CNN(18,36) + LSTM	0.5267	0.5621	0.5375	0	0.54

week - 2th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.4616	0.5185	0.4925	0	0	0.49	0
CNN(1,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(16,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(24,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(32,1) + LSTM	0.4616	0.5185	0.4925	0	0	0.49	0
CNN(36,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(40,1) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(36,18) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0
CNN(18,36) + LSTM	0.3959	0.4617	0.4925	0	0	0.49	0

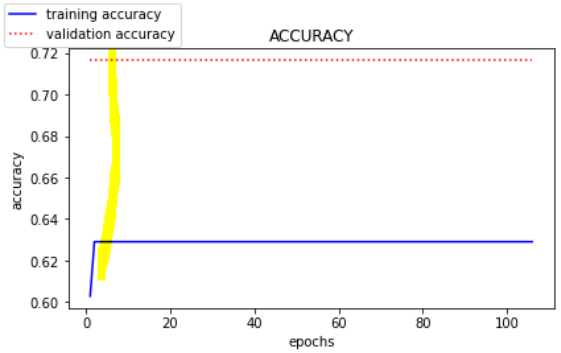
실험 결과

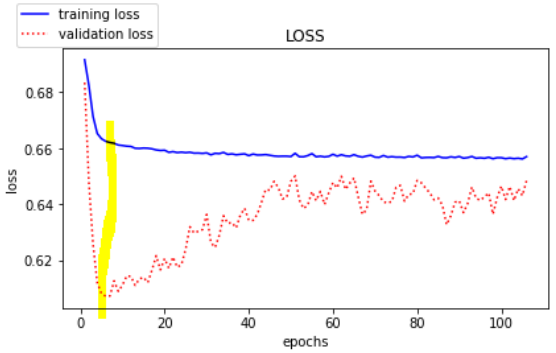
Month Test (binary classification) Result

S&P 500 : CNN(32, 1) + LSTM, CNN(36, 1) + LSTM

S&P 500데이터 Best Result Model의 Test set의 정확도는 0.5164, 0.6228이고,
Validation data와 train data의 정확도 또한 50% 이상입니다.
상승에 대한 정확도 86%, 79%입니다.

S&P 500 지수 데이터 그래프 특징은 우상향하는 그래프입니다.
이러한 그래프의 경우는 하락에 대한 특징보다는 상승에 대한 특징을 더욱 잘 교육할 수 있으며,
결국 상승의 경우가 하락의 경우보다 더 높은 정확도를 나타낼 수 있게 됩니다.
(모두 다 상승으로만 예측했을 때의 정확도 0.70%)






```
----------------------각 상황의 카운터----------------------------
Up_count :  189
Up_label_count =  448
Down_count =  450
Down_label_count =  191
----------------------각 상황의 예측 정확도----------------------------
Up evaluate rate =  0.8677248677248677
Down evaluate rate =  0.3688888888888889
```

실험 결과

Month Test (binary classification) Result

이는 주가 예측에 사용되는 딥러닝 모델의 성능 평가 지표로 단순 Test set의 정확도를 사용하는 경우에
평가에 대한 오류 발생할 수 있다는 것을 증명합니다.

또한 모델 중 위에서 언급하였듯 주가의 상승 및 하락을 예측하는 모델은 완벽하게 교육될 수 없기 때문에,

Test set의 정확도는 높지만 Train, validation data의 정확도가 낮은 경우도 존재합니다.

ex.

week - 2th	train	validation	test	상승	하락
LSTM	0.5312	0.4706	0.5493	0.37	0.56
CNN(1,1) + LSTM	0.5318	0.4706	0.5685	0	0.57

또한 해당 모델을 실제로 사용하는 데에 있어, 어떤 선택이 높은 정확도를 가지는지
또한 중요한 실험결과 지표 라는 것을 증명합니다.

실험 결과

Month Test (binary classification) Result

한국전력 Best Model: CNN(36,18) + LSTM

한국전력의 경우 Simple LSTM과 CNN(36,18) + LSTM 모델의 경우 좋은 성능을 보입니다.
그 중 Simple LSTM의 경우 실제 Test set의 정확도는 0.5462이지만, 하락의 정확도는 0.64입니다.

CNN(36,18) + LSTM 모델의 경우는 Test set의 정확도는 0.6039이지만 하락일 경우 0.63의 성능을 확인할 수 있습니다.

아시아나 Best Model : CNN(18,36) + LSTM

테스트 결과에서 상승의 예측률이 높은
(ex. Simple LSTM, CNN(1,1)+LSTM, CNN(16,1) + LSTM, CNN(24,1)) 모델들도 존재하였지만,
실제로 상승을 예측하는 횟수가 낮기 때문에 상승을 예측하는 결과를 제외하였습니다.
따라서 Test set의 정확도는 0.5332이며, 그 중 하락의 정확도는 66%입니다.

제외된 케이스 :

Model - 2th	train	validation	test	상승	하락
LSTM	0.542	0.6059	0.5691	1	0.55
CNN(1,1) + LSTM	0.5502	0.6059	0.5365	0.71	0.53
CNN(16,1) + LSTM	0.5461	0.6059	0.5571	0.87	0.55
CNN(24,1) + LSTM	0.5502	0.6059	0.5702	0.9	0.55
CNN(32,1) + LSTM	0.5488	0.6032	0.5365	0.54	0.54
CNN(36,1) + LSTM	0.562	0.6059	0.5441	0.54	0.54

실험 결과

Month Test (4 - th classification) Result

강한 상승 약한 상승 약한 하락 강한 하락으로 예측했을 경우, Train에 대해 어려움을 겪었습니다.

유일하게 다중 분류로 4가지의 예측이 나온 데이터가 한국 전력 데이터입니다.
모델은 총 3가지(ex. Simple LSTM, CNN(1,1) + LSTM, CNN(36, 18) + LSTM)이며,
만일 모든 예측을 약한 하락만 선택할 경우 54% 예측정확률이 나오게 됩니다.

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.2718	0.2815	0.2372	0.24	0.17	0.14	0.26
CNN(1,1) + LSTM	0.2724	0.3423	0.3341	0.31	0.28	0.22	0.44
CNN(16,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(24,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(32,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(36,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(40,1) + LSTM	0.4987	0.4488	0.5439	0	0	0.54	0
CNN(36,18) + LSTM	0.2718	0.2815	0.3101	0.25	0.18	0.31	0.34
CNN(18,36) + LSTM	0.4122	0.4167	0.5212	0	0	0.52	0

단순 하강 상승을 분류하는 모델보다
강한 상승, 약한 상승, 약한 하락, 강한 하락으로 분류하는 모델은 정확도가 낮은 것을 확인할 수 있습니다.

실험 결과

Month Test (4 - th classification) Result

S&P 500의 경우 강한 상승과 강한학습의 경우 예측이 불가능했고, 약한 상승과, 약한 하락으로만 예측되었습니다.
이 중 모델의 약한 상승의 정확도가 79% 및 77%이지만,
상승 및 하락의 분류모델 보다 성능이 좋지 않은 것을 확인 할 수 있었습니다.

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.5335	0.6533	0.6322	0	0.63	0	0
CNN(1,1) + LSTM	0.5335	0.6533	0.4304	0	0.77	0.28	0
CNN(16,1) + LSTM	0.5335	0.6533	0.4679	0	0.77	0.3	0
CNN(24,1) + LSTM	0.5335	0.6533	0.4304	0	0.68	0.31	0
CNN(32,1) + LSTM	0.5335	0.6533	0.6322	0	0.63	0	0
CNN(36,1) + LSTM	0.5335	0.6533	0.6103	0	0.62	0	0
CNN(40,1) + LSTM	0.5335	0.6553	0.3756	0	0.79	0.26	0
CNN(36,18) + LSTM	0.5335	0.6553	0.6322	0	0.63	0	0
CNN(18,36) + LSTM	0.5335	0.6553	0.6322	0	0.63	0	0

아시아나 주가 데이터의 경우도 S&P 처럼 이진 분류밖에 되지 않으며, 강한 상승, 강한 하락으로만 교육됩니다.

Model - 4th	train	validation	test	강한 상승	약한 상승	약한 하락	강한 하락
LSTM	0.3811	0.509	0.3787	0.33	0	0.42	0
CNN(1,1) + LSTM	0.3959	0.4617	0.3961	0.35	0	0.43	0
CNN(16,1) + LSTM	0.3938	0.4302	0.3874	0.52	0	0.38	0
CNN(24,1) + LSTM	0.3985	0.482	0.4	0.35	0	0.45	0
CNN(32,1) + LSTM	0.3962	0.536	0.3493	0.3	0	0.42	0
CNN(36,1) + LSTM	0.4044	0.5068	0.3351	0.28	0	0.42	0
CNN(40,1) + LSTM	0.3979	0.4932	0.3047	0.28	0	0.37	0
CNN(36,18) + LSTM	0.395	0.4887	0.3025	0.26	0	0.41	0
CNN(18,36) + LSTM	0.4477	0.4865	0.3645	0.21	0	0.39	0

결론

결론

본 논문의 첫번째 목표

CNN과 LSTM의 혼합 모델을 사용하여 주가의 상승 및 하락을 예측하는 모델은

성능적으로 좋은 예측력 가질 수 있는 것을 확인하였습니다.

하지만 다중 분류(강한 상승, 약한 상승, 약한 하락, 강한 하락)의 경우 제대로 학습이 어려운 것으로 확인됩니다.

또한 모델의 최상의 성능은 각 주가 종목마다 다르며,

CNN 필터의 개수에 따라서 달라지는 것을 확인할 수 있습니다.

따라서 이를 실제로 적용할 시 다른 주가 데이터를 분석하여, 최상의 결과를 내는 튜닝 값을 찾아내야 합니다.

결론

본 논문의 두번째 목표

딥러닝을 주가 예측 모델에 적용할 시 정확도를 평가할 시에는 Test data set의 정확도가 낮더라도

한 가지의 예측률이 높은 경우가 존재합니다.

한가지를 잘 예측하는 것 또한 **실제 매매전략**에 사용될 수 있으며,

따라서 주가의 등락을 예측하는 모델을 평가할 경우 필요한 새로운 지표로 사용되어야 합니다.

또한 **주가의 상승 및 하락을 예측하는 모델은 완벽하게 교육할 수 없으므로**,

Test data set의 정확도가 높더라도 Validation data 와 train data의 정확도 낮은 경우가 존재합니다.

따라서 Test data set의 정확도 뿐만 아닌 train set, Validation set의 정확도 또한 모델의 평가지표에 넣어야 합니다.

발전 가능성

발전 가능성

일주일 뒤의 주가를 예측하는 모델을 만들고 교육하기 위해서는,

해당 지난 데이터들이 이후 데이터와의 관계가 존재해야하므로,

결국 해당 모델을 평가하고자 하는 주가데이터의 특성이 중요합니다.

따라서 급변하는 특성을 가진 주가 데이터들을 가지고 테스트를 진행해도 좋을 것 같으며,

주가 데이터의 변동량만을 사용하여, 데이터 셋을 구성한다면 다른 결과를 만들 수 있을 것으로도 보입니다.

또한 주가에 대한 분석을 강한 상승과 강한 하락으로만 분류하는 모델을 만든다고 하고,

이를 직접적으로 테스트를 할 경우 평가 결과가 좋을 경우, 이 또한 주가의 오르내림을 예측할 수 있는

좋은 모델로 사용될 수 있을 듯 합니다.