**Coding과제 5**

목차

1. 주가nvidia RNN모델 소스코드
2. 실행 결과

학번 20140269 이름 고혁훈



* 과제의 목적

RNN으로 주가를 예측해본다.

Input sequence 길이에 따른 성능을 본다.

Lstm hidden unit과 layer 개수를 조절해본다.

* 배운 점

첫 번째로, 예측값이 실제 라벨값과 비슷하게 만들려고 hidden unit의 개수를 늘렸지만 곡선 형태만 비슷하게 나올 뿐 완전히 적합되지 않고 loss값이 줄어들다가 일정 정도에 수렴하는 것을 볼 수 있었다.

두 번째로, 그래서 lstm unit의 개수를 늘리지 않고 sequence의 개수를 늘렸더니 놀랍게도 큰 폭으로 loss값이 줄어들고 예측 곡선과 실제 label값이 잘 적합됨을 알 수 있었다.

세 번째로, lstm unit 대신에 layer를 하나 더 추가했더니 train 로스는 줄어들었지만 validation loss는 오히려 증가해서 train data에 과적합된다는 것을 깨달았다.

하지만 lstm 2개를 사용했을 때, unit도 줄이고 sequence도 같이 줄이니, lstm 1개에 sequence만 늘렸던 것에 비해 loss는 비교적 늘어났지만 전반적으로 곡선 적합도는 제일 안정적으로 나타났다.

종합적으로 정리하자면, 성능을 개선하고자 할 때, sequence부터 늘리고 그 다음에 hidden units의 개수를 loss가 크게 줄지 않는 지점까지 늘리는 것이 좋을 것 같다. 또한 layer 개수와 sequence 길이와 hidden unit을 다양하게 조절할 경우 어느 한쪽만 키우지 말고 각각 비례해서 키우는 것이 더 안정적인 모델을 만들 수 있다는 것을 알 수 있었다.

* 기본 소스코드

1) 임포트

from sklearn.preprocessing import MinMaxScaler

import numpy as np

import pandas as pd

import tensorflow as tf

import matplotlib.pyplot as plt

2) 데이터 load

data = pd.read\_csv('nvidia.csv')

# 필요없는 행을 삭제한다.

data = data.drop(['Adj Close','Date'],axis=1)

x = data

y = data['Volume']

3) 데이터 preprocess

# 데이터를 minmax로 normalize를 한다.

scaler = MinMaxScaler()

scaler.fit(x)

x = scaler.transform(x)

y = y.to\_numpy().reshape(-1,1)

scaler.fit(y)

y = scaler.transform(y)

# 하이퍼 파라미터를 정해준다.

timesteps = 7

data\_dim = 5

output\_dim = 1.

epoch = 150

# train 데이터를 만들고 split한다.

datax = []

datay = []

for i in range(0,len(y) - timesteps):

tempx = x[i:i+timesteps]

tempy = y[i+timesteps]

datax.append(tempx)

datay.append(tempy)

len(datax),len(datay)

#### split

tr\_size = int(len(datay)\*0.7)

trx,ttx = np.array(datax[:tr\_size]), np.array(datax[tr\_size:])

tr\_y,tt\_y = np.array(datay[:tr\_size]), np.array(datay[tr\_size:])

4) 모델생성

# layer는 되도록 많이 쌓아서 해볼 것이다. 그래서 차원에 맞게 model -

# layer를 자동생성해주는 코드를 만들었습니다.

layers = []

layers.append(Flatten(input\_shape=(28,28)))

temp = in\_shape

while(1):

temp = temp // (2\*\*2)

if temp < num\_class \* 2:

break

layer = tf.keras.layers.Dense(temp,activation='relu')

layers.append(layer)

layers.append(tf.keras.layers.Dense(num\_class,activation='softmax'))

# model loss와 optimizer, metric 장착

model = tf.keras.Sequential([tf.keras.layers.LSTM(10,input\_shape=(7,5)),tf.keras.layers.Dense(units=1,activation='linear')])

model.compile(optimizer='adam',loss='mse',metrics=[tf.keras.metrics.RootMeanSquaredError()])

history = model.fit(trx,tr\_y,epochs=150)

* 실행결과

1. 모델 summary

Model: "sequential"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

lstm (LSTM) (None, 10) 640

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 11

=================================================================

Total params: 651

Trainable params: 651

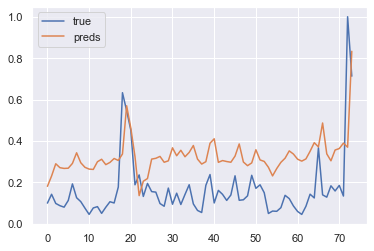
Non-trainable params: 0

1. epoch결과 일부

Epoch 145/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0136 - root\_mean\_squared\_error: 0.1168

1. fit 결과 도표(x축은 epoch)



1. 평가결과

loss: 0.0421 - root\_mean\_squared\_error: 0.2053

* 실행결과2(비교)

1) 모델 summary

1.1 hidden unit 10 -> 30

1.2 hidden unit 10 -> 50

1.3 hidden unit 10 -> 100

1.4 hidden unit 10 -> 200

2.1 sequence 7 -> 12

2.2 sequence 7 -> 20

2.3 sequence 7 -> 40

3.1 lstm 1개 -> 2개 hidden node with 200, 50

3.2 lstm 1개 -> 2개 hidden node with 100, 100

3.3 lstm 1개 -> 2개 hidden node with 50, 50 and sequence with 20

3.4 lstm 1개 -> 2개 hidden node with 50, 50 and sequence with 100

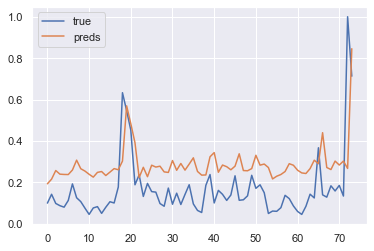
2) 평가 및 도표

1.1 hidden unit 10 -> 30

Epoch 150/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0132 - root\_mean\_squared\_error: 0.1149

3/3 [==============================] - 0s 1ms/step - loss: 0.0309 - root\_mean\_squared\_error: 0.1758

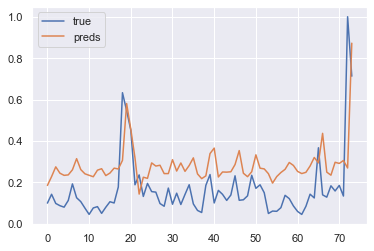


1.2 hidden unit 10 -> 50

Epoch 150/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0128 - root\_mean\_squared\_error: 0.1132

3/3 [==============================] - 0s 1ms/step - loss: 0.0301 - root\_mean\_squared\_error: 0.1735

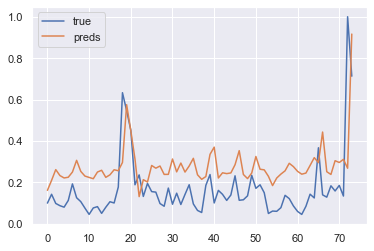


1.3 hidden unit 10 -> 100

Epoch 132/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0128 - root\_mean\_squared\_error: 0.1133

3/3 [==============================] - 0s 1ms/step - loss: 0.0292 - root\_mean\_squared\_error: 0.1710

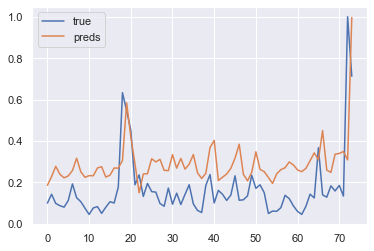


1.4 hidden unit 10 -> 200

Epoch 134/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0127 - root\_mean\_squared\_error: 0.1127

3/3 [==============================] - 0s 1ms/step - loss: 0.0326 - root\_mean\_squared\_error: 0.1806

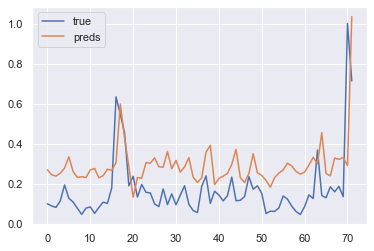


2.1 sequence 7 -> 12

Epoch 150/150

6/6 [==============================] - 0s 2ms/step - loss: 0.0126 - root\_mean\_squared\_error: 0.1125

3/3 [==============================] - 0s 1ms/step - loss: 0.0335 - root\_mean\_squared\_error: 0.1830

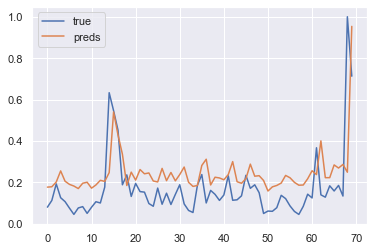


2.2 sequence 7 -> 20

Epoch 143/150

6/6 [==============================] - 0s 3ms/step - loss: 0.0136 - root\_mean\_squared\_error: 0.1165

3/3 [==============================] - 0s 1ms/step - loss: 0.0227 - root\_mean\_squared\_error: 0.1506

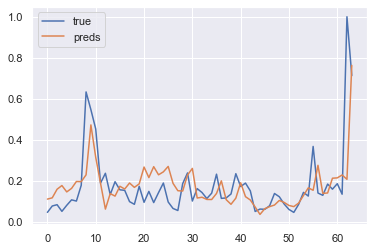


2.3 sequence 7 -> 40

Epoch 136/150

5/5 [==============================] - 0s 3ms/step - loss: 0.0106 - root\_mean\_squared\_error: 0.1028

2/2 [==============================] - 0s 2ms/step - loss: 0.0188 - root\_mean\_squared\_error: 0.1370

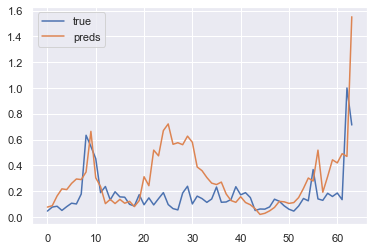


3.1 lstm 1개 -> 2개 hidden node with 200, 200

Epoch 149/150

5/5 [==============================] - 0s 6ms/step - loss: 0.0101 - root\_mean\_squared\_error: 0.1006

2/2 [==============================] - 0s 2ms/step - loss: 0.0619 - root\_mean\_squared\_error: 0.2488

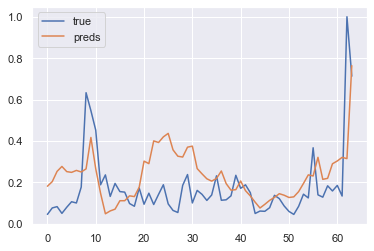


3.2 lstm 1개 -> 2개 hidden node with 200, 50 and sequence with 40

Epoch 148/150

5/5 [==============================] - 0s 5ms/step - loss: 0.0086 - root\_mean\_squared\_error: 0.0925

2/2 [==============================] - 0s 2ms/step - loss: 0.0284 - root\_mean\_squared\_error: 0.1684

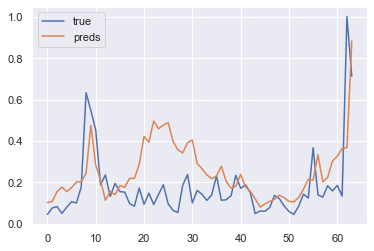


3.3 lstm 1개 -> 2개 hidden node with 100, 100 and sequence with 40

Epoch 148/150

5/5 [==============================] - 0s 4ms/step - loss: 0.0096 - root\_mean\_squared\_error: 0.0982

2/2 [==============================] - 0s 1ms/step - loss: 0.0316 - root\_mean\_squared\_error: 0.1778

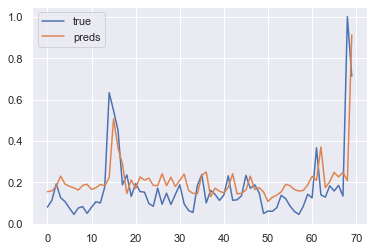


3.4 lstm 1개 -> 2개 hidden node with 50, 50 and sequence with 20

Epoch 147/150

6/6 [==============================] - 0s 3ms/step - loss: 0.0134 - root\_mean\_squared\_error: 0.1156

3/3 [==============================] - 0s 2ms/step - loss: 0.0191 - root\_mean\_squared\_error: 0.1380



3.5 lstm 1개 -> 2개 hidden node with 100, 100 and sequence with 20

Epoch 136/150

6/6 [==============================] - 0s 3ms/step - loss: 0.0134 - root\_mean\_squared\_error: 0.1158

3/3 [==============================] - 0s 2ms/step - loss: 0.0195 - root\_mean\_squared\_error: 0.1398

