**Detection of anomaly Stock Price Based on Time Series Deep Learning Models**

**Abstract:**

anomaly detection is a critical task for financial market, investors, and regulatory authorities, where conventional methods employ rule-based models. With the development of machine learning and deep learning techniques, it becomes more promising to detect anomalous trading behaviors from data. Here we present a deep learning model based on time series LSTM model to detect anomalous behaviors in Chinese stock market. The model is composed of 1dConv-LSTM neurons, which can predict time series stock price data from historical data. We analyzed the price of 14 stocks variations ranging from 2015/01/05 to 2019/12/31 and used univariate and multivariate time series models to generate MAE less than 4.0 consistently. The proposed method improved MSE to 0.0171 on validation datasets.Our model successfully predicts the anomalous price behaviors of ‘601318’ stock in the range of 2019-02-13. Our method provide an automatic way of predict anomaly stock price behavior in Chinese stock market.

Keywords: Finance, DL, LSTM, Time Series, Anomalous Stock Price

**1. Introduction**:

The Luckin Coffee fraud scandal in the last few months has been spread worldwide. According to the internal investigation, it shows that the fabrication of sales began in April 2019, which included inflating costs and expenses by almost $200 million, as well as booking $300 million in false revenue. After the announcement, Luckin’s stock has slumped 32%. Such a scandal made it harder for other Chinese companies to debut in the United States and thus lose a huge amount of US potential investors due to the untrustworthiness of Chinese companies. Besides, the event magnifies the defect of the Chinese stock market that requires the mechanism to oversee anomalous behaviors of stocks and judge which is a fraud for regulatory authorities. Only if the Chinese authorities establish the mechanism to identify anomalous stock price fluctuations and investigate hoax can investments be secure, and the stock market be more stable.

Various detection systems to monitor abnormal stock price changes have been developed. Previously, most of the detection methods rely on a prediction-based method or rule-based method. [prediction-based method or rule-based method explain more]

Since mid-1997, the National Association of Securities Dealers (NASD) in the United States developed Advanced Detection System (ADS) that has been used to monitor trades and quotations in the NASDAQ stock market.The ADS uses two pattern matches to detect abnormal behaviors. The system relies on a rule matcher, which detects predefined suspicious behaviors, and a time-sequence matcher, which looks for temporal relationships between events that exist in a potential violation pattern.

According to Time Series Contextual Anomaly Detection for Detecting Market Manipulation in Stock Market, the author proposed Contextual Anomaly Detection (CAD) method, which aims to use unsupervised way to exploit the behavior of similar time series to predict the expected values. The biggest problem of such a method is its lack of precision and accuracy of the result due to unknown and unlabelled data. It generally has a recall about 7%.1

Currently, the automatic anomoly detection system in China is still under development.

[what we did] We propose a DL method → improve precision and acc. [三人集体修改]

DL method what other did → examples/algos [1-2 para] [Alina]

Summary our advance. %, detect event [Ruofan]

**2. Dataset:**

We analyzed **13** stocks in total, including companies that become listed in Shanghai and Shenzhen Stock exchanges. We gathered these data from dataset YouKuang2, using codes such as “DataAPI.MktEqudGet(secID=u600448.XSHG",beginDate=u"20150101",endDate=u"",field=u"",pandas="1")”, and saving data in the form of CSV for further analysis. Besides, we recorded the Open, Close Price, Turnover Value, Deal Amount, etc of each stock from 2015 to 2020.

Data is splited into training and testing data according to the ‘tradeDate’, where it starts from 2015/01/05 to 2019/12/31. The trading date range is 1219 days, where we set days = 1000 as the split. The first 1000 days are set to training data, and the last 219 days are validating data.

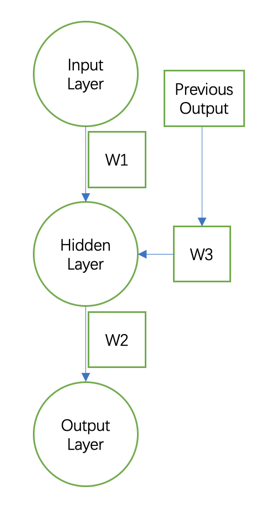
Table 1. Example columns of dataset. [Alina 重新画]

**3. Methodology:**

**3.1 Models**

The model is tested for 13 random selected stock tickers, where we slice the data using 1D conv method and used the data to feed into the LSTM model that can predict the time series of ‘StockPrice’.

**3.1.1 Recurrent Neural Network (RNN):**RNN is better than traditional neural networks in areas that require people to consider previous results when calculating current situations. The biggest advantage of RNN is that it can memorize previous information, not just containing the input layer and the output layer. The figure is shown as below:

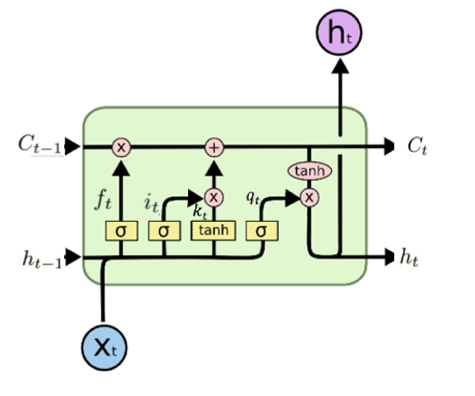


Fig[1]: The Inner Structure of RNN

Here, W1, W2, W3 are all weight vectors. Specifically, W1 is the weight vector given to the information passing from the input layer to the hidden layer, W2 is the weight vector from the hidden layer to the output layer and W3 is the weight vector from the previous output to the hidden layer. The output contains not only the input information, but also the previous output of the neural network.

**3.1.2 Long Short Term Memory Networks (LSTM):**

It is true that RNN can remember previous outputs, however, the memory cannot last for too long -- for long conveying distances, RNN does not behave well. To solve this problem, we use LSTM here, a special kind of RNN. The following graph (Fig 2) demonstrates the inner structure of this model:

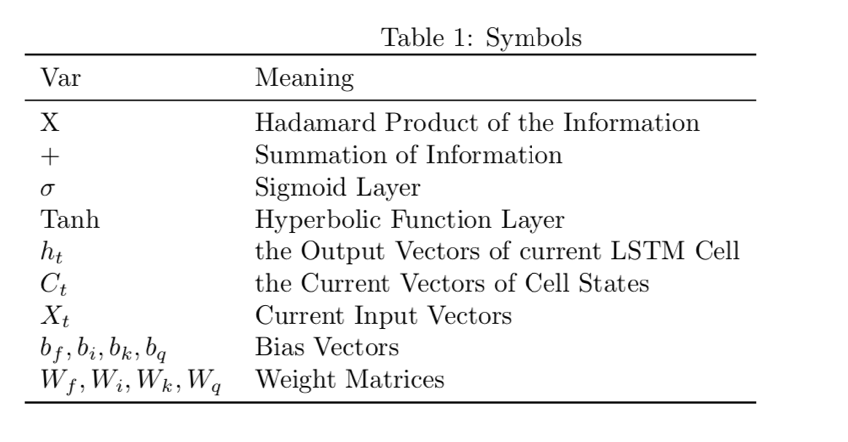


Fig[2]: The Inner Structure of LSTM Unit (Picture Originally from

<https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. This is the modified version)

The yellow rectangle represents Neural Network Layer, the pink circle represents pointwise operation and the arrows mainly stand for vectors.

Table 1 listed variables that will be deployed later in the formula:



There are four steps to pass through LSTM:

The first step is to determine which part of the input vector should be removed. Let denotes the activation value of forget gates at time , then depends mainly on and . The sigmoid layer forces the output , where no information will pass if :

(1)

The second step is to add additional information to the previous matrix, which consists of two parts: using a sigmoid layer to help decide which information will pass and computing the candidate vector which passes Tanh layer. Let denote the activation function of the input gate and the candidate vector, respectively:

(2)

(3)

The third step is to combine the first and the second step by multiplying the information and their corresponding weights (activation values):

(4)

The last part is to calculate the output of this LSTM model . Similar to the second step, denote to be the activation value of output:

(5)

(6)

**3.1.3 Conv1D-LSTM:**

Conv1D-LSTM is used to derive features from datasets, especially in time series data. As is shown in Figure 1. The univariant and multivariant data was feed into the Conv1D layer to slice into readable length data for LSTM inputs. Using Conv1D layer can ‘chop’ time series data into the correct format to feed into LSTM. Here we window each 30 days as a block of data with a batch size equals to 32. The Conv1D uses 10 filters with a kernel size equal to 5 and a stride step 1. After this layer, sequential data is partitioned into various types of LSTM plus dense neural layers.3

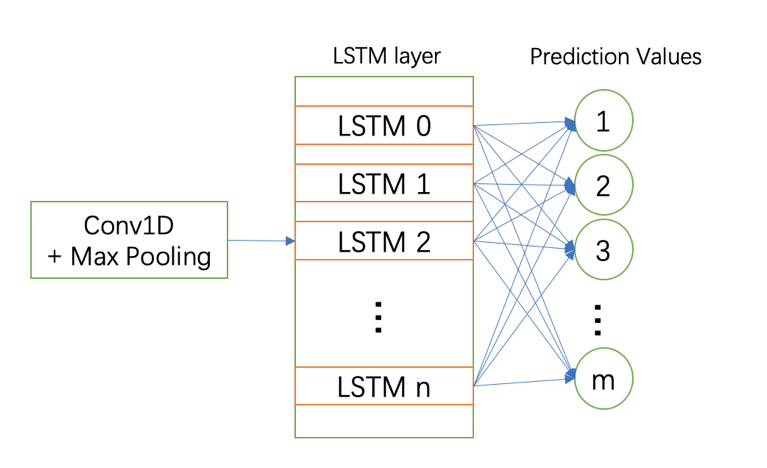


Fig1. Structure of Conv1D-LSTM model prediction.4

We demonstrate how the OpenPrice looks like in a stock, ‘MinSheng’ , as shown in Figure 2. The volume and open price can be used to construct multivariant datasets.

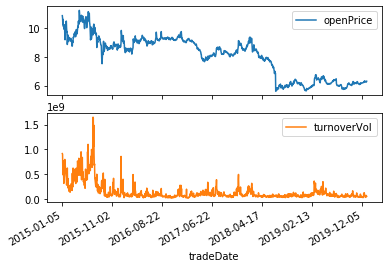
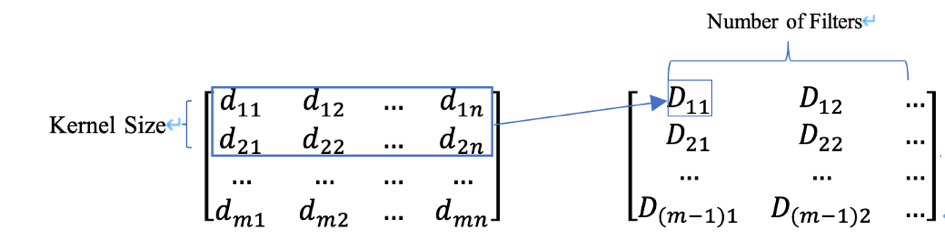
****

Fig2. Time-Series data of MinSheng Stock showing the open price and turnover Volume.

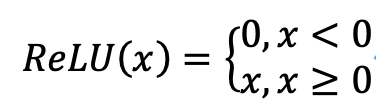
By sliding kernels along the preprocessed data, we can get feature maps. (Shown in Fig[4])

****

Fig[4]: The Process of Getting Feature Map

**3.1.4 Non-Linear Activation Layer:**

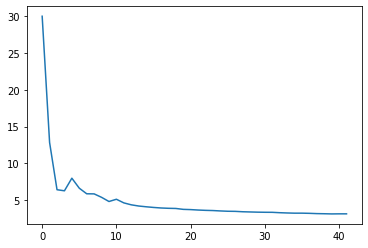
Often, non-linear activation layer is employed after convolutional layer and fully-connected layer, since non-linear operation can help keep every change made in linear operation. There are many types of non-linear function, such as Logistic Function, Hyperbolic Tangent Function, Rectified Linear Function (ReLU), etc. In this article, we use ReLU as activation function:

**** (7)

which makes calculation speed faster than the other functions.

**3.2 Workflow**

First, we divided all the data into training dataset and testing dataset. As there are about 1219 data in a stock, thus we divided them into 1000 training data and 219 testing data. Then, after plotting the whole data to see the stock’s trend, we train the training data through one Conv1D-LSTM layer, two LSTM layers and three Dense layers with “ReLU” as activation functions and thus figure out the best learning rate, an indicator used to measure the step size of each iteration of the model towards the loss function. The figure below reflects the exponential decline of the learning rate when training the data:



The Change of Learning Rate in Stock ‘Beidouxingtong’

The lowest y-value of the graph is the best learning rate we want. Later, by considering “open price” and “turnover volume” as two main factors to influence stock price, we used Multi-step model to build the corresponding neural network -- two LSTM layers and a Dense layer with “ReLU” activation function. By comparing the predicted values with the actual values, anomalous stock prices can be found.

**4. Result**

**4.1 Model Performance**

In order to evaluate the model performance in training and val/test dataset, we use MAE and MSE as our metrics. We use the first 1000 days open price and volume as multivariant data and use the validation data set of 1001 to end for model tuning purpose. Figure 5 shows a predicted example of stock ‘603718’ using the Conv1D-LSTM model.



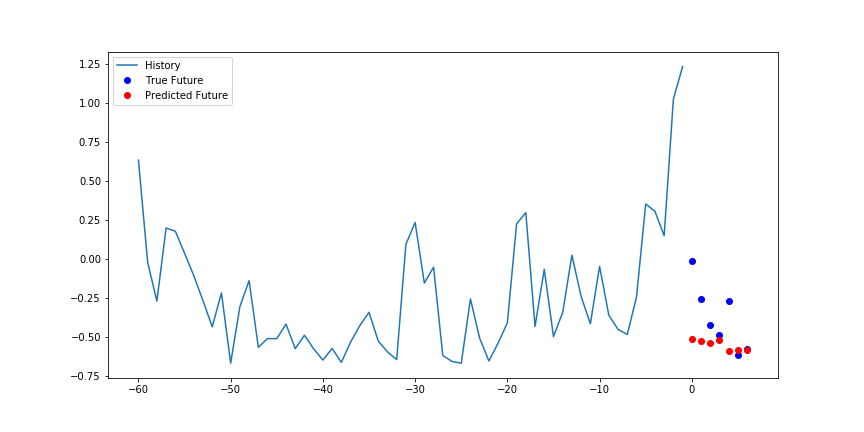


Figure 5 Model prediction on validation dataset of 603718’

The naïve baseline model by shifting the date of one day gives a MAE more than 6.0 on all datasets. The example data ‘603718’ has a MAE less than 4.5 where its prediction on validation captures the trend of open price.

**4.2 Predicted anomaly behavior**

We randomly picked 13 stocks and used the range after 1250 to test if the model performance is statically different with previous period performances. After training, the model can predict unseen data from previous datasets. We used this model prediction as a predictor of possible anomaly behavior. If the deviation of MAE is larger than 30% on the training set, we mark the predicted period problematic. Using this strategy, we identify possible anomaly behavior of ‘601318’ in the period of test as shown in Figure 6.

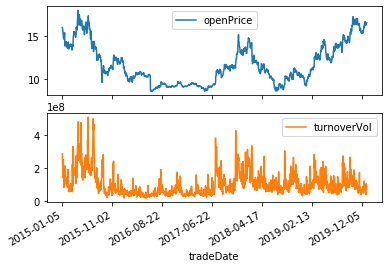
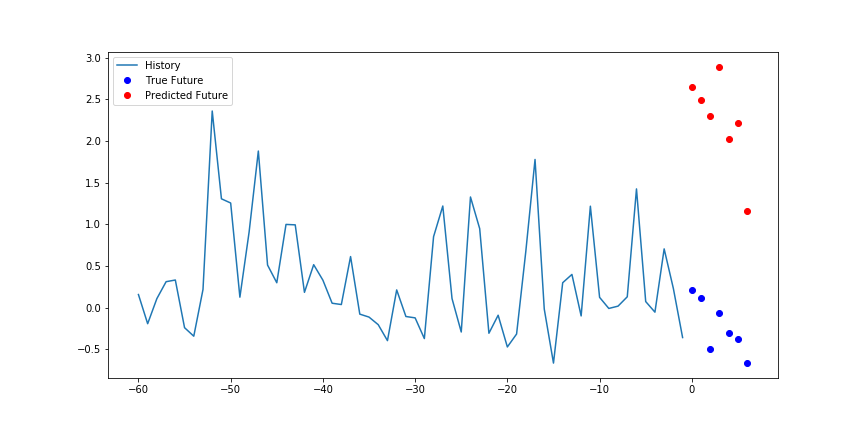


Figure 6 Anomaly detection of 601318 in a 7 days period.

We can see that on 2019-02-13 where is set to 0 in the right, we observed abnormal volume and price jump that differs a lot from previous trends. This shows a possible point of anomaly behavior.

**5. Conclusion**

In this paper, we proposed a deep learning model that can learn the historical trend from previous traded stocks prices and make predictions automatically. This model can lower the MAE to a low level, beyond which anomaly behavior of stock is possible. We demonstrated that in 2019-02-13 period of 7 days, it is possible to have high volume and price jump. This deep learning method can be an effective predictor of anomaly stock price behavior in China.

**References:**

1. Golmohammadi, K. & Zaiane, O. R. Time series contextual anomaly detection for detecting market manipulation in stock market. in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)* 1–10 (IEEE, 2015). doi:10.1109/DSAA.2015.7344856

2. QUER. *https://uqer.datayes.com/* (2020).

3. Fischer, T. & Krauss, C. Deep learning with long short-term memory networks for financial market predictions. *Eur. J. Oper. Res.* **270**, 654–669 (2018).

4. Understanding LSTM Networks. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

**Appendix**

# -\*- coding: utf-8 -\*-

"""

Created on Sun Aug 30 10:27:47 2020

@author: Jason, Alina, Ruofan

"""

# general imports

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import os

import glob

# print(tf.\_\_version\_\_)

# Define functions

def plot\_series(time, series, format="-", start=0, end=None):

plt.plot(time[start:end], series[start:end], format)

plt.xlabel("Time")

plt.ylabel("Value")

plt.grid(False)

def windowed\_dataset(series, window\_size, batch\_size, shuffle\_buffer):

series = tf.expand\_dims(series, axis=-1)

ds = tf.data.Dataset.from\_tensor\_slices(series)

ds = ds.window(window\_size + 1, shift=1, drop\_remainder=True)

ds = ds.flat\_map(lambda w: w.batch(window\_size + 1))

ds = ds.shuffle(shuffle\_buffer)

ds = ds.map(lambda w: (w[:-1], w[1:]))

return ds.batch(batch\_size).prefetch(1)

def model\_forecast(model, series, window\_size):

ds = tf.data.Dataset.from\_tensor\_slices(series)

ds = ds.window(window\_size, shift=1, drop\_remainder=True)

ds = ds.flat\_map(lambda w: w.batch(window\_size))

ds = ds.batch(32).prefetch(1)

forecast = model.predict(ds)

return forecast

def create\_time\_steps(length):

return list(range(-length, 0))

def show\_plot(plot\_data, delta, title):

labels = ['History', 'True Future', 'Model Prediction']

marker = ['.-', 'rx', 'go']

time\_steps = create\_time\_steps(plot\_data[0].shape[0])

if delta:

future = delta

else:

future = 0

plt.title(title)

for i, x in enumerate(plot\_data):

if i:

plt.plot(future, plot\_data[i], marker[i], markersize=10,

label=labels[i])

else:

plt.plot(time\_steps, plot\_data[i].flatten(), marker[i], label=labels[i])

plt.legend()

plt.xlim([time\_steps[0], (future+5)\*2])

plt.xlabel('Time-Step')

return plt

def listfiles(path):

files = glob.glob(path+'\\\*')

return files

def readfile(file,col\_to\_read):

df = pd.read\_csv(file)

df = df[col\_to\_read]

return df

def run\_md(file):

df = readfile(file,col\_to\_read)

ss = file.split('\\')[-1][:-4]

series = df['openPrice'].to\_numpy()

time = np.array(df.index)

plt.figure(figsize=(10, 6))

plot\_series(time, series)

split\_time = 1000

time\_train = time[:split\_time]

x\_train = series[:split\_time]

time\_valid = time[split\_time:]

x\_valid = series[split\_time:]

window\_size = 30

batch\_size = 32

shuffle\_buffer\_size = 1000

BATCH\_SIZE = 32

BUFFER\_SIZE = 10000

# model 1

tf.keras.backend.clear\_session()

tf.random.set\_seed(51)

np.random.seed(51)

train\_set = windowed\_dataset(x\_train, window\_size=30, batch\_size=32, shuffle\_buffer=shuffle\_buffer\_size)

model = tf.keras.models.Sequential([

tf.keras.layers.Conv1D(filters=10, kernel\_size=5,

strides=1, padding="causal",

activation="relu",

input\_shape=[None, 1]),

tf.keras.layers.LSTM(10, return\_sequences=True),

# tf.keras.layers.LSTM(30, return\_sequences=True),

tf.keras.layers.Dense(30, activation="relu"),

tf.keras.layers.Dense(10, activation="relu"),

tf.keras.layers.Dense(1),

tf.keras.layers.Lambda(lambda x: x \* 2400)

])

lr\_schedule = tf.keras.optimizers.schedules.ExponentialDecay(

1e-6,

decay\_steps=20,

decay\_rate=0.96,

staircase=True)

optimizer = tf.keras.optimizers.SGD(learning\_rate=lr\_schedule, momentum=0.9)

model.compile(loss=tf.keras.losses.Huber(),

optimizer=optimizer,

metrics=["mae"])

history = model.fit(train\_set,epochs=50,verbose=0)

# run focast

rnn\_forecast = model\_forecast(model, series[..., np.newaxis], window\_size)

rnn\_forecast = rnn\_forecast[split\_time - window\_size:-1,-1, 0]

plt.figure(figsize=(10, 6))

plot\_series(time\_valid, x\_valid)

plot\_series(time\_valid, rnn\_forecast)

plt.legend()

plt.savefig('plt\\plt\_hist\_%s'%ss)

plt.figure()

plt.plot(history.history['loss'])

plt.savefig('plt\\plt\_%s.png'%ss)

def multivariate\_data(dataset, target, start\_index, end\_index, history\_size,

target\_size, step, single\_step=False):

data = []

labels = []

start\_index = start\_index + history\_size

if end\_index is None:

end\_index = len(dataset) - target\_size

for i in range(start\_index, end\_index):

indices = range(i-history\_size, i, step)

data.append(dataset[indices])

if single\_step:

labels.append(target[i+target\_size])

else:

labels.append(target[i:i+target\_size])

return np.array(data), np.array(labels)

#$$$$$$$$$$$$$$$$$$$$$$$$$

def multi\_md(file):

df = pd.read\_csv(file)

ss = file.split('\\')[-1][:-4]

TRAIN\_SPLIT = 1000

features\_considered = ['openPrice', 'turnoverVol']

features = df[features\_considered]

features.index = df['tradeDate']

dataset = features.values

data\_mean = dataset[:TRAIN\_SPLIT].mean(axis=0)

data\_std = dataset[:TRAIN\_SPLIT].std(axis=0)

dataset = (dataset-data\_mean)/data\_std

# Single step model

def multivariate\_data(dataset, target, start\_index, end\_index, history\_size,

target\_size, step, single\_step=False):

data = []

labels = []

start\_index = start\_index + history\_size

if end\_index is None:

end\_index = len(dataset) - target\_size

for i in range(start\_index, end\_index):

indices = range(i-history\_size, i, step)

data.append(dataset[indices])

if single\_step:

labels.append(target[i+target\_size])

else:

labels.append(target[i:i+target\_size])

return np.array(data), np.array(labels)

past\_history = 60

future\_target = 7

STEP = 1

# multistep model

future\_target = 7

x\_train\_multi, y\_train\_multi = multivariate\_data(dataset, dataset[:, 1], 0,

TRAIN\_SPLIT, past\_history,

future\_target, STEP)

x\_val\_multi, y\_val\_multi = multivariate\_data(dataset, dataset[:, 1],

TRAIN\_SPLIT, None, past\_history,

future\_target, STEP)

train\_data\_multi = tf.data.Dataset.from\_tensor\_slices((x\_train\_multi, y\_train\_multi))

train\_data\_multi = train\_data\_multi.cache().shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE).repeat()

val\_data\_multi = tf.data.Dataset.from\_tensor\_slices((x\_val\_multi, y\_val\_multi))

val\_data\_multi = val\_data\_multi.batch(BATCH\_SIZE).repeat()

def rev(ds):

return ds\*data\_std+data\_mean

def multi\_step\_plot(history, true\_future, prediction,ss):

plt.figure(figsize=(12, 6))

num\_in = create\_time\_steps(len(history))

num\_out = len(true\_future)

plt.plot(num\_in, np.array(history[:, 1]), label='History')

plt.plot(np.arange(num\_out)/STEP, np.array(true\_future), 'bo',

label='True Future')

if prediction.any():

plt.plot(np.arange(num\_out)/STEP, np.array(prediction), 'ro',

label='Predicted Future')

plt.legend(loc='upper left')

plt.savefig('plt\_pred\_%s.png'%ss)

plt.show()

print('Success plt %s' %ss)

# multistep model

multi\_step\_model = tf.keras.models.Sequential()

multi\_step\_model.add(tf.keras.layers.LSTM(16,

return\_sequences=True,

input\_shape=x\_train\_multi.shape[-2:]))

multi\_step\_model.add(tf.keras.layers.LSTM(16, activation='relu'))

multi\_step\_model.add(tf.keras.layers.Dense(7))

multi\_step\_model.compile(optimizer=tf.keras.optimizers.RMSprop(clipvalue=1.0), loss='mse')

EVALUATION\_INTERVAL = 200

EPOCHS = 20

multi\_step\_history = multi\_step\_model.fit(train\_data\_multi, epochs=EPOCHS,

steps\_per\_epoch=EVALUATION\_INTERVAL,

validation\_data=val\_data\_multi,

validation\_steps=50,verbose=1)

def plot\_train\_history(history, title,ss):

loss = history.history['loss']

val\_loss = history.history['val\_loss']

epochs = range(len(loss))

plt.figure()

plt.plot(epochs, loss, 'b', label='Training loss')

plt.plot(epochs, val\_loss, 'r', label='Validation loss')

plt.title(title)

plt.ylim([0,5])

plt.legend()

plt.savefig('plt\\plt\_pred7\_hist\_%s.png'%ss)

plt.show()

plot\_train\_history(multi\_step\_history,

'Single Step Training and validation loss',ss)

for x, y in val\_data\_multi.take(1):

multi\_step\_plot(x[0], y[0], multi\_step\_model.predict(x)[0],ss)

#$$$$$$$$$$$$$$$$$$$$$$$$$

# =============================================================================

# This is the section of main file execution

# =============================================================================

global col\_to\_read

col\_to\_read = ['openPrice','turnoverVol']

files = listfiles('D:\\gits\\stock\\code\\data')

file = files[-1]

window\_size = 30

batch\_size = 32

shuffle\_buffer\_size = 1000

BATCH\_SIZE = 32

BUFFER\_SIZE = 10000

for file in files[5:]:

print('run %s file' %file)

# run\_md(file)

multi\_md(file)