武汉大学计算机学院

课程论文

课程名称 商务智能

专业年级班级 软件工程 2016级 卓越2班

姓名 张大珂

学号 2016302580213

学期 2018-2019 学年 第一 学期

成绩 任课教师签名

武汉大学计算机学院
本科生课程设计报告

基于 LSTM 的黄金期货价格预测
Prediction of Gold Future Prices Based on LSTM

专 业 名 称：	软件工程
课 程 名 称：	商务智能
指 导 教 师：	朱卫平 副教授
学 生 学 号：	2016302580213
学 生 姓 名：	张大珂

二〇一八年十二月

郑 重 声 明

Abstract

A method to make predictions about the prices of Gold Futures based on LSTM (Long Short Term Memory) is proposed in this report. As a practice of Predictive Analytics in the field of Business Intelligence, this method including data preprocessing, neural network training, testing and evaluation. In the experiment based on PaddlePaddle, with the advantages of LSTM, the proposed method performs well in the predictions of Gold T+D Prices of China.

Keywords: Business Intelligence, Recurrent Neural Network, Long Short Term Memory, Portfolio Optimization

Contents

1 Background

An increasing number of methods based on neural network have been proposed to make predictions in the stock market. Meanwhile, prices of futures like gold and silver are a form of times series data like stocks. It is natural to try applying these methods on the analysis of prices of futures. Classical methods depend on artificial identification of patterns and design of algorithms, which will cost a lot of time and leave little room to improve.

Time Series like stock prices and future prices rely heavily on historical prices. However, CNN (Convolutional Neural Networks) have limited capability for temporal memory of previous prices. Recurrent neural networks have a cycle which feeds activations from the previous time step back in as an input and influences the activations of the current time step. Furthermore, LSTM networks are a specific type of recurrent neural network which overcomes some of the problems of recurrent networks.

During my research, I find prices of several futures demonstrate strong similarity. As shown in Figure 1.1, I draw three time series in one chart where I shift the lines to make the comparison more straightforward since they have different ranges. Therefore, in the proposed method, there are four time series that help make predictions about Gold T+D to improve accuracy.

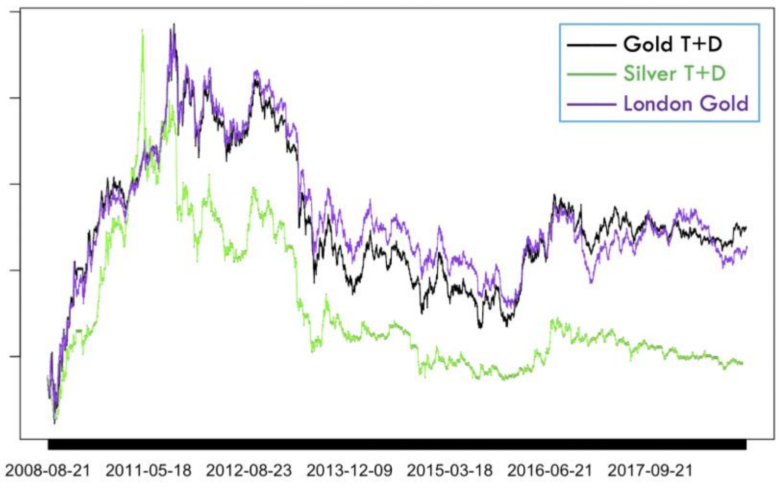


Figure 1.1: Prices of Gold T+D, Silver T+D and London Gold

2 Requirement Analysis

2.1 Business Needs

Nowadays, machine learning has come to play an integral role in many parts of the financial ecosystem, from portfolio management and algorithmic trading, to fraud detection and loan/insurance underwriting. [1] Predictions about prices of futures are of great importance to individual investors and investment companies. There are numerous factors that affect the investment market, so it is becoming more and more difficult and inaccurate for artificial predictions. Meanwhile, since the market changes rapidly, the predictions should be made in time with satisfied accuracy. Therefore, there is a strong desire for new techniques to make reliable predictions.

2.2 Solutions Prospects

In addition to traditional econometric models, the development of machine learning algorithms marks a new era for modeling complex financial time series. What's more, neural networks do not require too much background knowledge of economy. The training process is like a black box, while the result is quite satisfying. Besides, the training time of the proposed method is relatively short, so the predictions could be made in time for investors to make investment decisions. With more and more related research, I'm convinced that machine learning will bring revolution to the financial market.

2.3 Limitations

The predictions of the proposed method are not very accurate when the gold market become unstable, which means the price of gold goes up or drops down dramatically. Actually, this is the inherent limitations of machine learning algorithms since they can only make predictions on patterns that they learn from the previous data. Once there is a new factor interfering the market, the proposed method will have poor performance.

3 Approaches

3.1 Time Series

Time series are one of the most common data types encountered in finance, and so time-series analysis is one of the most widely used traditional approaches in finance and economics. [1]

The definition of time series is a set of observations on the values that a variable takes at different times. Note that this variable can be represented by multiple dimensions, like a vector at different times. And time series analysis is a statistical technique that deals with time series data. Time series are like arrays except that the indexes are of date type. And there are functions among popular programming languages like R and Python which can be used directly and efficiently.

Recently, financial time-series modeling has become one of the most popular topics in the field of application of machine learning. And there have been numerous successful methods and theories that have great effects on the financial market.

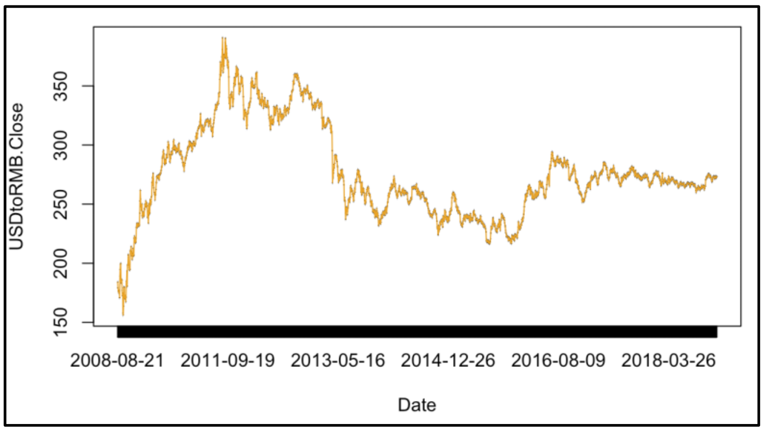


Figure 3.1 Visualization of time series

3.2 Long Short Term Memory

In this report, time series under consideration, like stock prices and future prices, depend on historical prices, which means previous information matters in making predictions. However, traditional neural networks can’t make use of that information.

Thus, recurrent neural networks are raised to address this issue. As is shown in Figure 3.2, there are loops within RNNs, allowing information to persist.

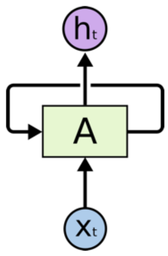


Figure 3.2: Loops in Recurrent Neural Networks [3]

In some cases, we only need to use recent information, which means the gap between the relevant information and the current place is small. RNNs can learn to use this kind of past information. But in many other cases, we need more context and the gap could become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information. [3] These cases have long term dependencies.

Hochreiter and Schmidhuber introduced Long Short Term Memory (LSTM) networks in 1997, which is a special kind of RNN, capable of learning long-term dependencies. All RNNs have the form of a chain of repeating modules of neural network. But the repeating module of LSTMs has a different structure. Instead of one single neural network layer, there are four layers in LSTMs, interacting in a very special way.

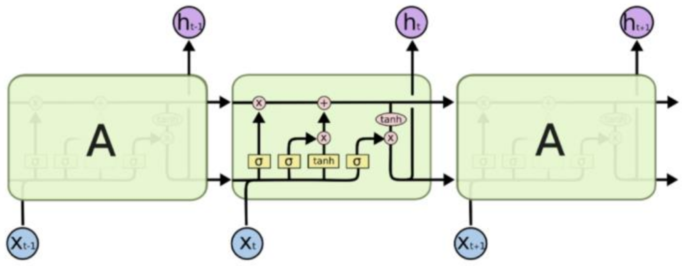


Figure 3.3: The repeating module in an LSTM containing four interacting layers [3]

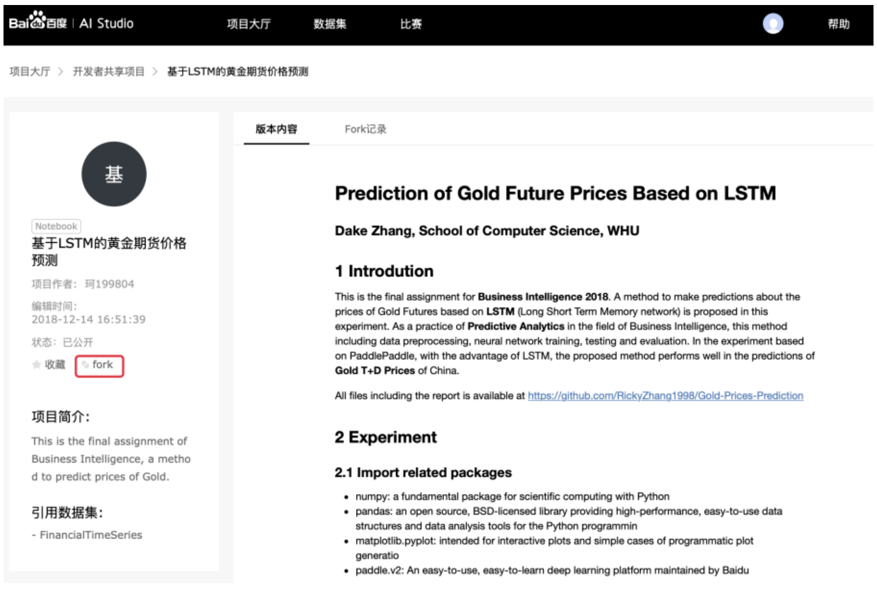
As is shown in Figure 3.3, each line carries an entire vector, from the output of one node to the inputs of others. The pink circles represent pointwise operations, while the yellow boxes are learned neural network layers. Lines merging denote concatenation, while a line forking denotes its content being copied and the copies going to different locations. [3]

Therefore, the proposed method in this report is based on LSTM to make prediction about future prices which have long term dependencies.

4 Experiment

4.1 Environment

This experiment is on the AI studio (http://aistudio.baidu.com/). The source code is available at http://aistudio.baidu.com/?_=1538223849065#/projectdetail/34678 (please log in first).



基于LSTM的黄金期货价格预测

项目作者：珂199804

编辑时间：2018-12-14 16:51:39

状态：已公开

项目简介：

This is the final assignment of Business Intelligence, a method to predict prices of Gold.

引用数据集：

- FinancialTimeSeries

Prediction of Gold Future Prices Based on LSTM

Dake Zhang, School of Computer Science, WHU

1 Introdution

This is the final assignment for **Business Intelligence 2018**. A method to make predictions about the prices of Gold Futures based on **LSTM** (Long Short Term Memory network) is proposed in this experiment. As a practice of **Predictive Analytics** in the field of Business Intelligence, this method including data preprocessing, neural network training, testing and evaluation. In the experiment based on PaddlePaddle, with the advantage of LSTM, the proposed method performs well in the predictions of **Gold T+D Prices** of China.

All files including the report is available at https://github.com/RickyZhang1998/Gold-Prices-Prediction

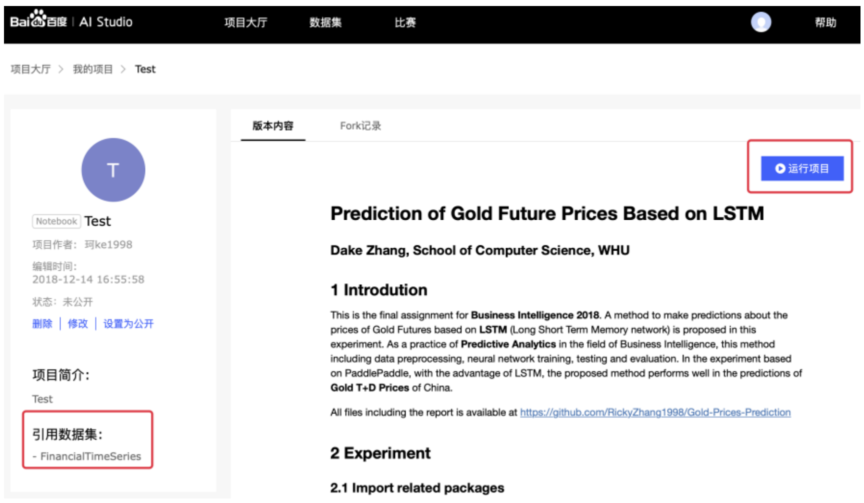
2 Experiment

2.1 Import related packages

- numpy: a fundamental package for scientific computing with Python
- pandas: an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programmin
- matplotlib.pyplot: intended for interactive plots and simple cases of programmatic plot generatio
- paddle.v2: An easy-to-use, easy-to-learn deep learning platform maintained by Baidu

Figure 4.1: Fork the project at the website mentioned above

As is shown in Figure 4.1, the project can be forked since I have made the project public. After forking the project, the data set is automatically added and the forked project can run immediately as is shown in Figure 4.2.



Test

项目作者：珂ke1998

编辑时间：2018-12-14 16:55:58

状态：未公开

项目简介：

Test

引用数据集：

- FinancialTimeSeries

Prediction of Gold Future Prices Based on LSTM

Dake Zhang, School of Computer Science, WHU

1 Introdution

This is the final assignment for **Business Intelligence 2018**. A method to make predictions about the prices of Gold Futures based on **LSTM** (Long Short Term Memory network) is proposed in this experiment. As a practice of **Predictive Analytics** in the field of Business Intelligence, this method including data preprocessing, neural network training, testing and evaluation. In the experiment based on PaddlePaddle, with the advantage of LSTM, the proposed method performs well in the predictions of **Gold T+D Prices** of China.

All files including the report is available at https://github.com/RickyZhang1998/Gold-Prices-Prediction

2 Experiment

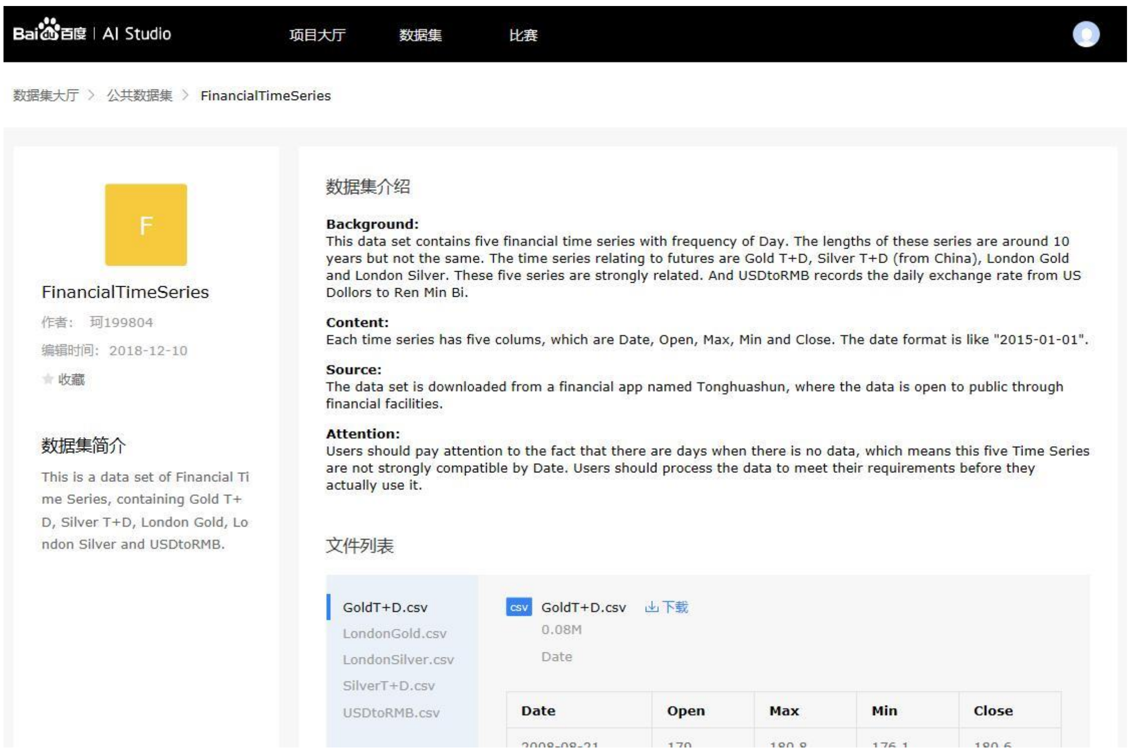
2.1 Import related packages

Figure 4.2: The forked project

In this empirical study, I use several packages and frameworks. Numpy is a fundamental package for scientific computing with Python. Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming. Package “matplotlib.pyplot” is intended for interactive plots and simple cases of programmatic plot generation. PaddlePaddle is an easy-to-use, easy-to-learn deep learning platform maintained by Baidu.

4.2 Data Set

The data set used in this experiment is downloaded from the application of Tonghuashun (http://www.10jqka.com.cn). The data set has been uploaded to AI Studio and can be viewed in this website http://aistudio.baidu.com/aistudio/#/datasetDetail/2233 (please log in AI Studio first), as is shown in Figure 4.3.



FinancialTimeSeries

作者： 珂199804

编辑时间： 2018-12-10

收藏

数据集简介

This is a data set of Financial Time Series, containing Gold T+D, Silver T+D, London Gold, London Silver and USDtoRMB.

数据集介绍

Background:
This data set contains five financial time series with frequency of Day. The lengths of these series are around 10 years but not the same. The time series relating to futures are Gold T+D, Silver T+D (from China), London Gold and London Silver. These five series are strongly related. And USDtoRMB records the daily exchange rate from US Dollors to Ren Min Bi.

Content:
Each time series has five colums, which are Date, Open, Max, Min and Close. The date format is like "2015-01-01".

Source:
The data set is downloaded from a financial app named Tonghuashun, where the data is open to public through financial facilities.

Attention:
Users should pay attention to the fact that there are days when there is no data, which means this five Time Series are not strongly compatible by Date. Users should process the data to meet their requirements before they actually use it.

文件列表

GoldT+D.csv
LondonGold.csv
LondonSilver.csv
SilverT+D.csv
USDtoRMB.csv

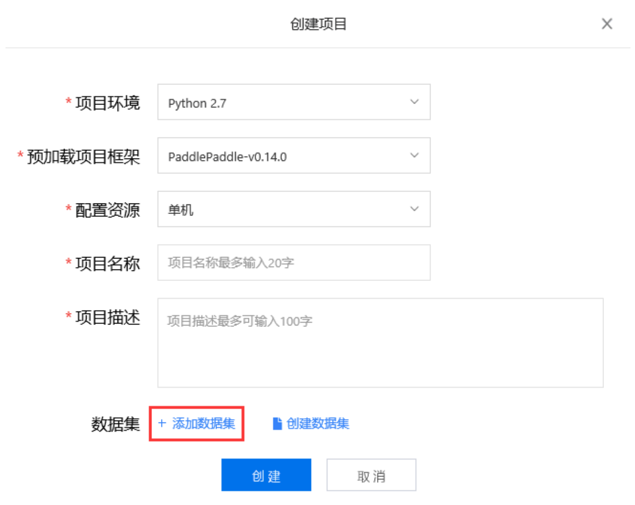
csv GoldT+D.csv 下载
0.08M
Date

Date	Open	Max	Min	Close
2008-08-21	170	180.8	176.1	180.6

Figure 4.3: Data set in this experiment in AI Studio

And the data set can be added either in the creation of a project (Figure 4.4) or in the modification of the created project (Figure 4.5).

Besides, the data set is also included in the package with path “/Data/TimeSeries”.


创建项目

* 项目环境: Python 2.7

* 预加载项目框架: PaddlePaddle-v0.14.0

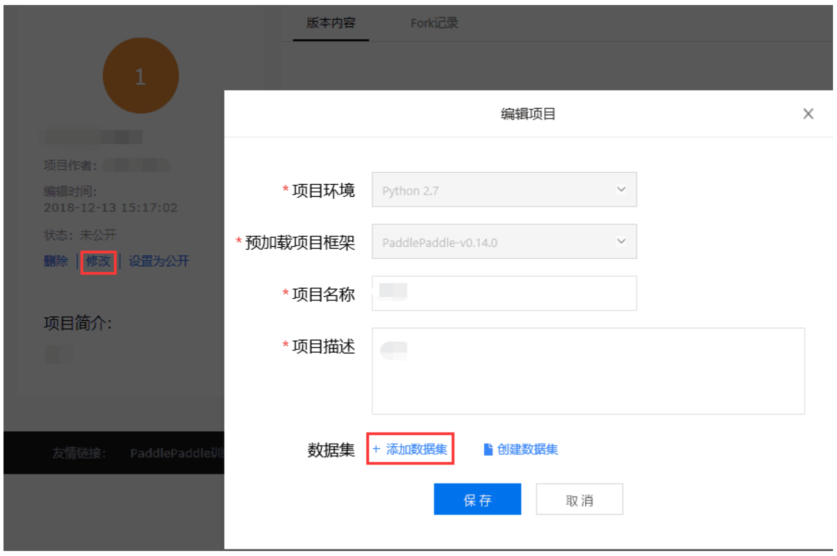
* 配置资源: 单机

* 项目名称: 项目名称最多输入20字

* 项目描述: 项目描述最多可输入100字

数据集: + 添加数据集 | 创建数据集

Figure 4.4: Add data set in the creation of a project



[redacted]

项目作者: [redacted]

编辑时间:
2018-12-13 15:17:02

状态: 未公开

项目简介:

[redacted]

友情链接: PaddlePaddle训

编辑项目

* 项目环境: Python 2.7

* 预加载项目框架: PaddlePaddle-v0.14.0

* 项目名称: [redacted]

* 项目描述: [redacted]

数据集: + 添加数据集 | 创建数据集

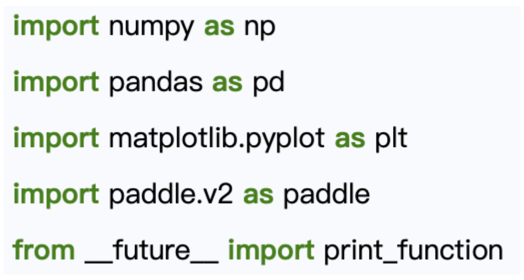
Figure 4.5: Add data set in the modification of the project

The main time series analyzed in this experiment is “GoldT+D.csv”, which records the gold prices in China between 2008-08-21 and 2018-12-06. Note that the daily records are not consecutive, because there is no trade on weekends and legal holidays. And the other three time series are used in making predictions about Gold T+D prices in China. They are in the files “SilverT+D.csv”, “LondonGold.csv”, and “LondonSilver.csv” respectively. Each time series has five columns, which are Date, Open, Max, Min and Close. As is mentioned before, these four time series are strongly related. Also note that the four time series in this data set are not fully corresponding by Date. We leave “USDtoRMB.csv” for further experiment.

4.3 Experiment Procedure

The purpose of this experiment is to make predictions of gold prices in China using LSTM. There are descriptions alongside each code block on the AI Studio website. In this section, we only focus on the explanation of crucial steps.

4.3.1 Import related packages



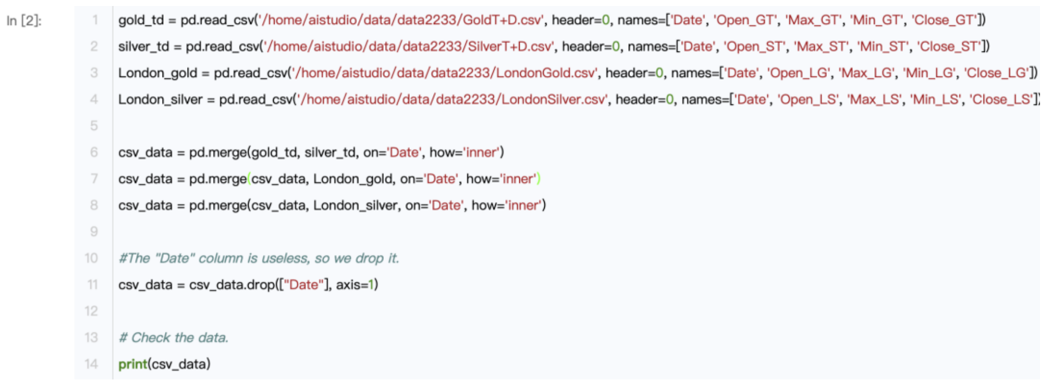
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import paddle.v2 as paddle
from __future__ import print_function
```

Figure 4.6: Import related packages illustrated in 4.1

4.3.2 Preprocess the data

A. Load time series from csv files and merge them

As is shown in Figure 4.7, in this experiment, we use four time series to make predictions about Gold T+D Open Prices. Note that the four time series are not fully corresponding by the date. So we choose to merge them in "inner" mode.

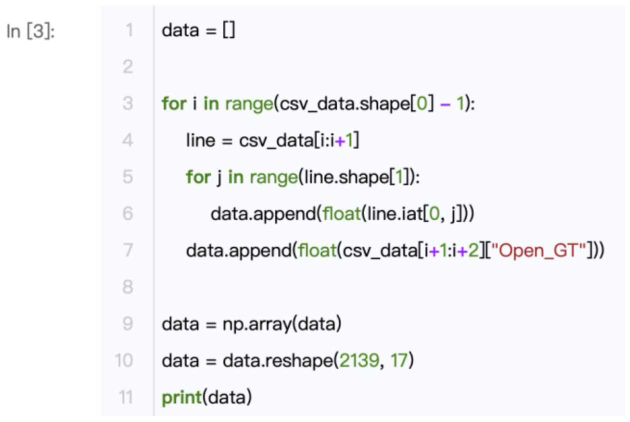


```
In [2]:
gold_td = pd.read_csv('/home/aistudio/data/data2233/GoldT+D.csv', header=0, names=['Date', 'Open_GT', 'Max_GT', 'Min_GT', 'Close_GT'])
silver_td = pd.read_csv('/home/aistudio/data/data2233/SilverT+D.csv', header=0, names=['Date', 'Open_ST', 'Max_ST', 'Min_ST', 'Close_ST'])
London_gold = pd.read_csv('/home/aistudio/data/data2233/LondonGold.csv', header=0, names=['Date', 'Open_LG', 'Max_LG', 'Min_LG', 'Close_LG'])
London_silver = pd.read_csv('/home/aistudio/data/data2233/LondonSilver.csv', header=0, names=['Date', 'Open_LS', 'Max_LS', 'Min_LS', 'Close_LS']

csv_data = pd.merge(gold_td, silver_td, on='Date', how='inner')
csv_data = pd.merge(csv_data, London_gold, on='Date', how='inner')
csv_data = pd.merge(csv_data, London_silver, on='Date', how='inner')

#The "Date" column is useless, so we drop it.
csv_data = csv_data.drop(["Date"], axis=1)

# Check the data.
print(csv_data)
```

Figure 4.7: Code block that loads time series and merges them

B. Change the form of data

As is shown in Figure 4.8, we firstly transfer csv_data of type "DataFrame" into array and then add the corresponding label, namely the open price of the next day. After that, we reshape the array into size 2139 × 17.

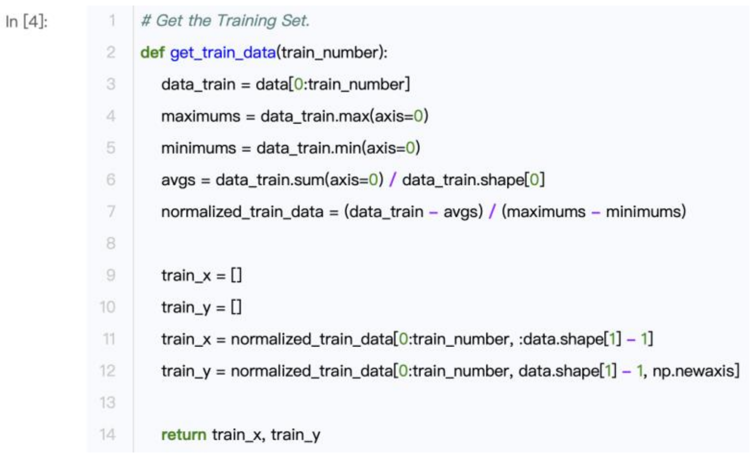


```
In [3]:
data = []

for i in range(csv_data.shape[0] - 1):
    line = csv_data[i:i+1]
    for j in range(line.shape[1]):
        data.append(float(line.iat[0, j]))
    data.append(float(csv_data[i+1:i+2]["Open_GT"]))

data = np.array(data)
data = data.reshape(2139, 17)
print(data)
```

Figure 4.8: Code block that changes the form of data

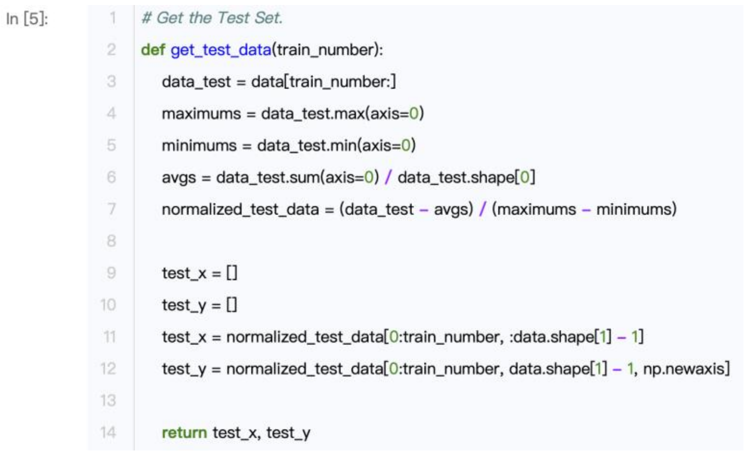
C. Define functions to get the training set and test set

As is shown in Figure 4.9 and 4.10, we use ratio 7:3 to separate the training set and the test set. Namely, we choose the first 1497 records as training set and the last 642 records as test set. And then we apply normalization to each feature/column to improve the accuracy and speed up the convergence of the model.



```
In [4]:
# Get the Training Set.
def get_train_data(train_number):
    data_train = data[0:train_number]
    maximums = data_train.max(axis=0)
    minimums = data_train.min(axis=0)
    avgs = data_train.sum(axis=0) / data_train.shape[0]
    normalized_train_data = (data_train - avgs) / (maximums - minimums)

    train_x = []
    train_y = []
    train_x = normalized_train_data[0:train_number, :data.shape[1] - 1]
    train_y = normalized_train_data[0:train_number, data.shape[1] - 1, np.newaxis]

    return train_x, train_y
```

Figure 4.9: Code block that gets the training set

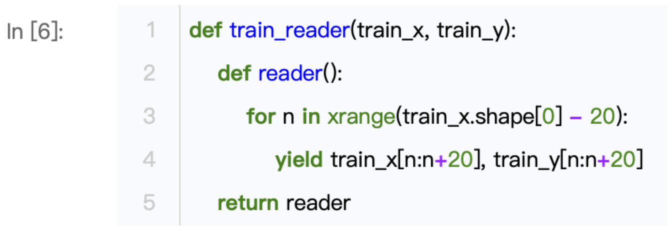


```
In [5]:
# Get the Test Set.
def get_test_data(train_number):
    data_test = data[train_number:]
    maximums = data_test.max(axis=0)
    minimums = data_test.min(axis=0)
    avgs = data_test.sum(axis=0) / data_test.shape[0]
    normalized_test_data = (data_test - avgs) / (maximums - minimums)

    test_x = []
    test_y = []
    test_x = normalized_test_data[0:train_number, :data.shape[1] - 1]
    test_y = normalized_test_data[0:train_number, data.shape[1] - 1, np.newaxis]

    return test_x, test_y
```

Figure 4.10: Code block that gets the test set

4.3.3 Define the reader

As is shown in Figure 4.11, we define a function as a reader to read Training Set and Test Set each time. Within the function, we use keyword “yield” to make reader() a Generator. In this way, we can reduce the pressure on memory. Since we sometimes cannot load the whole data set into memory at once which will be a great waste of memory, this generator will just get the data in need. In this case, we get 20 records every time.



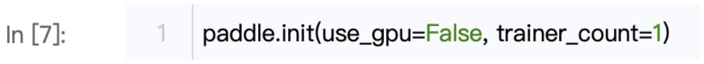
```
In [6]:
def train_reader(train_x, train_y):
    def reader():
        for n in xrange(train_x.shape[0] - 20):
            yield train_x[n:n+20], train_y[n:n+20]
    return reader
```

Figure 4.11: Code block that defines the reader

4.3.4 Design the LSTM model

A. Initialize the PaddlePaddle

In this case, we don’t use GPU to do the training and we set the trainer to be 1.

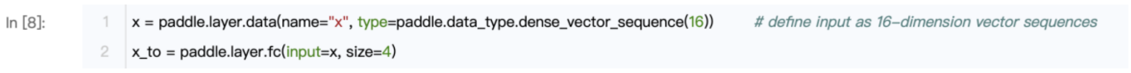


```
In [7]:
paddle.init(use_gpu=False, trainer_count=1)
```

Figure 4.12: Code block that initializes the PaddlePaddle

B. Define the input layer

We use time series as input and define the input as 16-dimension vector sequences which the 17th dimension is the label.

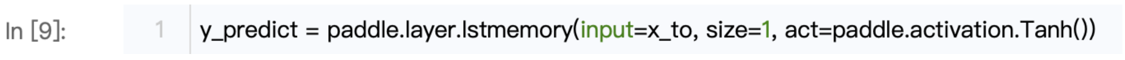


```
In [8]:
x = paddle.layer.data(name="x", type=paddle.data_type.dense_vector_sequence(16))      # define input as 16–dimension vector sequences
x_to = paddle.layer.fc(input=x, size=4)
```

Figure 4.13: Code block that defines the input layer

C. Define the output layer

We use function “paddle.layer.lstmmemory()” to define a LSTM layer. The input is “x_to”, which is defined above. The number of neuron is set to be 1. And the activation function is defined to be “Tanh()”.

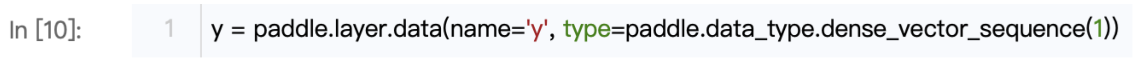


```
In [9]:
y_predict = paddle.layer.lstmemory(input=x_to, size=1, act=paddle.activation.Tanh())
```

Figure 4.14: Code block that defines the output layer

D. Define the label layer

This step is similar to the step that defines the input layer, except that we define the label to be 1 dimension value sequences.

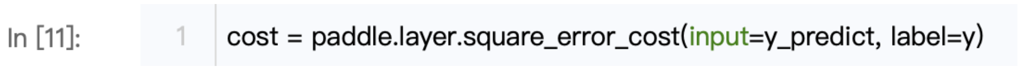


```
In [10]:    y = paddle.layer.data(name='y', type=paddle.data_type.dense_vector_sequence(1))
```

Figure 4.14: Code block that defines the label layer

E. Define the cost function

In this case, we use MSE (Mean Square Error) function to be the cost function to calculate the gradient and to refine the parameters.

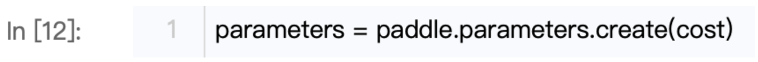


```
In [11]:    cost = paddle.layer.square_error_cost(input=y_predict, label=y)
```

Figure 4.15: Code block that defines the cost function

F. Create parameters

Function “paddle.parameters.create()” is used to create and initialize parameters. In this case, we use cost function defined above to create and initialize parameters.

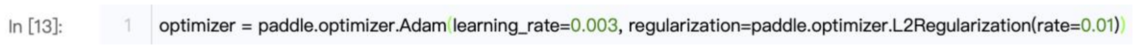


```
In [12]:    parameters = paddle.parameters.create(cost)
```

Figure 4.16: Code block that creates parameters

G. Create the optimizer

In this case, we use Adam optimizer provided by “paddle.optimizer.Adam()” as the optimizer. We set the learning rate to be 0.003.

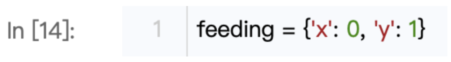


```
In [13]:    optimizer = paddle.optimizer.Adam(learning_rate=0.003, regularization=paddle.optimizer.L2Regularization(rate=0.01))
```

Figure 4.17: Code block that creates the optimizer

H. Define the map feeding

This feeding is a map from the name of the layer to the index of the array to input data while training.

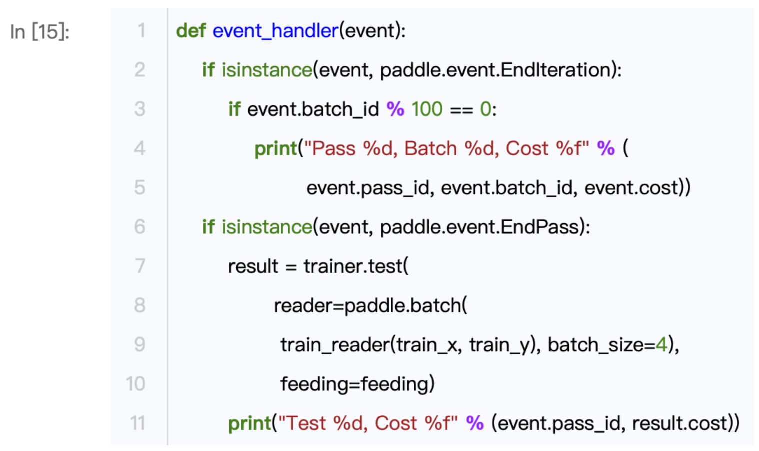


```
In [14]:    feeding = {'x': 0, 'y': 1}
```

Figure 4.18: Code block that defines the map feeding

I. Define the event handler

This event handler is defined to inform us the learning process.



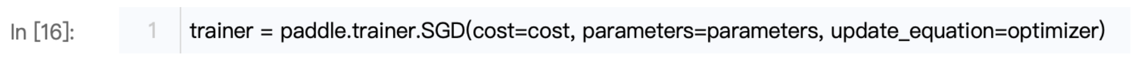
```
In [15]:
def event_handler(event):
    if isinstance(event, paddle.event.EndIteration):
        if event.batch_id % 100 == 0:
            print("Pass %d, Batch %d, Cost %f" % (
                    event.pass_id, event.batch_id, event.cost))
    if isinstance(event, paddle.event.EndPass):
        result = trainer.test(
            reader=paddle.batch(
             train_reader(train_x, train_y), batch_size=4),
             feeding=feeding)
        print("Test %d, Cost %f" % (event.pass_id, result.cost))
```

Figure 4.19: Code block that defines the event handler

4.3.5 Start training

A. Create a trainer

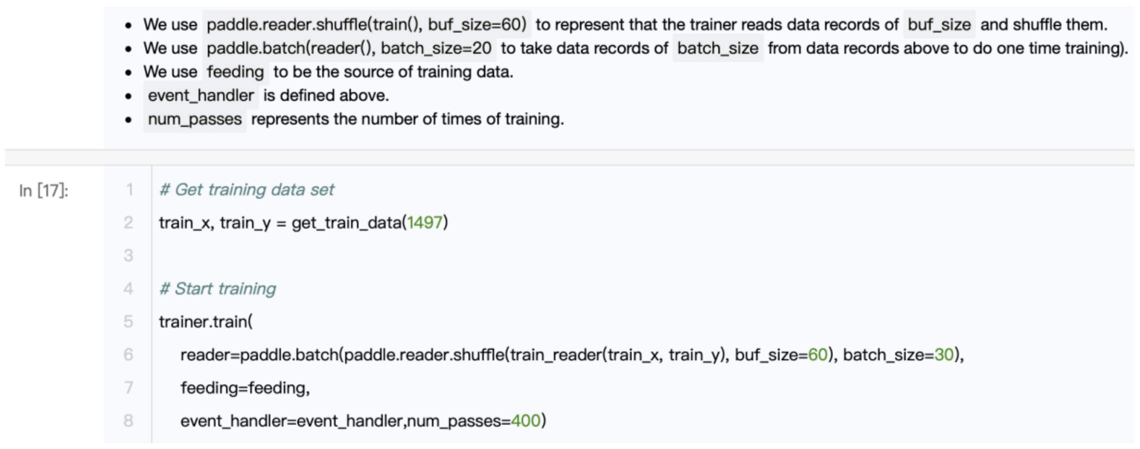
We use “paddle.trainer.SGD()” to define a Stochastic Gradient Descent trainer with cost function, parameters and optimizer all defined above.



```
In [16]:
trainer = paddle.trainer.SGD(cost=cost, parameters=parameters, update_equation=optimizer)
```

Figure 4.20: Code block that defines a trainer

B. Set training parameters and start training



- We use `paddle.reader.shuffle(train(), buf_size=60)` to represent that the trainer reads data records of `buf_size` and shuffle them.
- We use `paddle.batch(reader(), batch_size=20` to take data records of `batch_size` from data records above to do one time training).
- We use `feeding` to be the source of training data.
- `event_handler` is defined above.
- `num_passes` represents the number of times of training.

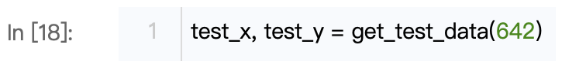
```
In [17]:
# Get training data set
train_x, train_y = get_train_data(1497)

# Start training
trainer.train(
    reader=paddle.batch(paddle.reader.shuffle(train_reader(train_x, train_y), buf_size=60), batch_size=30),
    feeding=feeding,
    event_handler=event_handler,num_passes=400)
```

Figure 4.21: Code block that sets training parameters and starts training

4.3.6 Make predictions and visualize the results

We have finished the training process. And we can use the model to make predictions and evaluate the model.

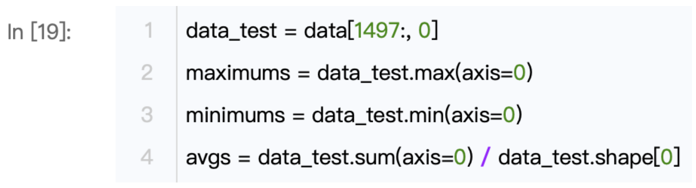
A. Get the test set



```
In [18]:  test_x, test_y = get_test_data(642)
```

Figure 4.22: Code block that gets the test set

B. Preprocess the test set

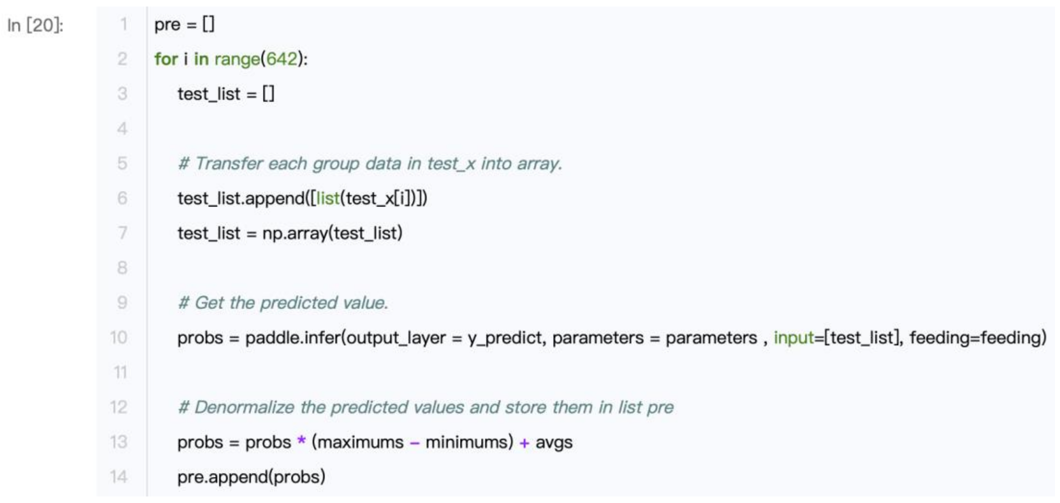


```
In [19]:  data_test = data[1497:, 0]
          maximums = data_test.max(axis=0)
          minimums = data_test.min(axis=0)
          avgs = data_test.sum(axis=0) / data_test.shape[0]
```

Figure 4.22: Code block that preprocesses the test set using normalization

C. Start testing

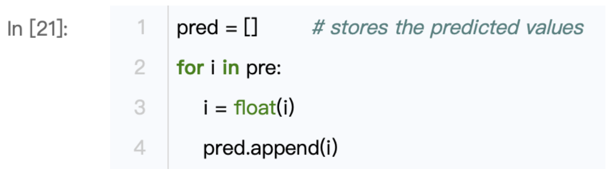
We use predict function “paddle.infer()” to make predictions.



```
In [20]:  pre = []
          for i in range(642):
              test_list = []

              # Transfer each group data in test_x into array.
              test_list.append([list(test_x[i])])
              test_list = np.array(test_list)

              # Get the predicted value.
              probs = paddle.infer(output_layer = y_predict, parameters = parameters , input=[test_list], feeding=feeding)

              # Denormalize the predicted values and store them in list pre
              probs = probs * (maximums - minimums) + avgs
              pre.append(probs)
```

Figure 4.22: Code block that starts testing

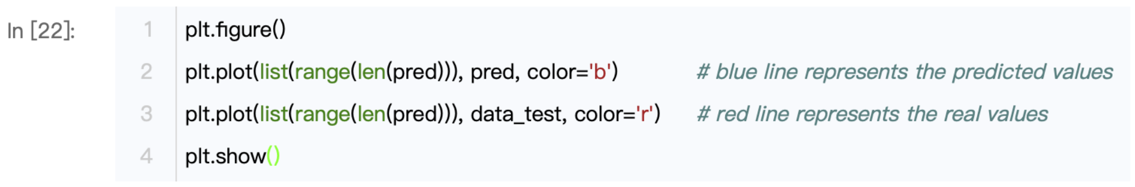
D. Use another list to store results for visualization



```
In [21]:  pred = []      # stores the predicted values
          for i in pre:
              i = float(i)
              pred.append(i)
```

Figure 4.23: Code block that uses another list to store results for visualization

E. Visualize the data



```
In [22]:  plt.figure()
          plt.plot(list(range(len(pred))), pred, color='b')        # blue line represents the predicted values
          plt.plot(list(range(len(pred))), data_test, color='r')   # red line represents the real values
          plt.show()
```

Figure 4.24: Code block that visualizes the data

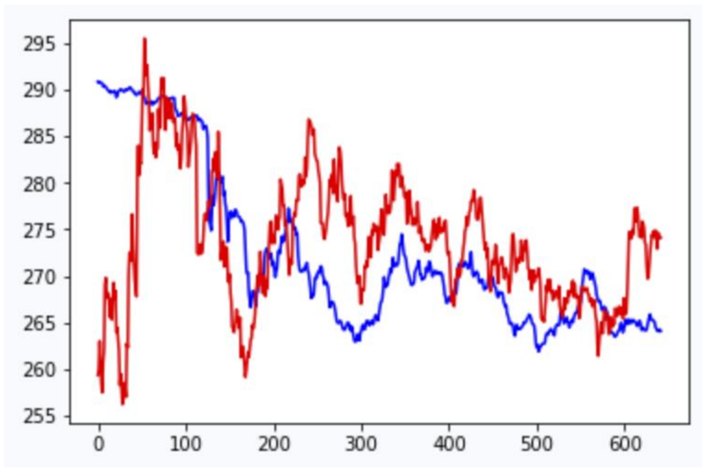
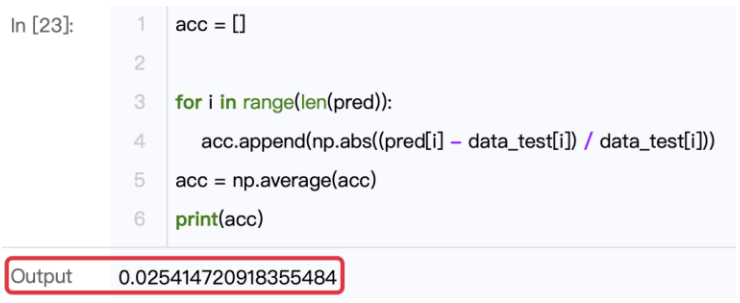


Figure 4.25: The prediction result of this experiment

F. Evaluate the model

We use Average Error to represent the accuracy of the model.



In [23]:
```
acc = []

for i in range(len(pred)):
    acc.append(np.abs((pred[i] - data_test[i]) / data_test[i]))
acc = np.average(acc)
print(acc)
```

Output
```
0.025414720918355484
```

Figure 4.26: Code block that evaluates the model and the result

5 Discussion

As is shown in Figure 4.26, the trained model performed well in the predictions with an average error of 2.54%. In the Figure 4.25, we can see except the first 50 points, the predicted line and the real line is quite similar. In most cases, the difference on y-axis is within 5 yuan. The reason why we might think the two lines are not very similar is that this graph is drawn in relatively narrow y-axis range which enlarges the difference between two lines.

And I think this experiment can be improved further. Taking the first 50 points as an example, when the gold market turns into an unstable stage, the prediction could be extremely inaccurate. Taking my recent scientific research into consideration, TDA (Topological Data Analysis) can detect this kind of chaos ahead of time. However, this method is not mature since the accuracy of it is not very satisfying. More work can be done in this field.

Acknowledgements

I would like to thank Professor Zhu for providing me with the fundamental knowledge of Business Intelligence.

I also thank developers of AI Studio and PaddlePaddle, who provided a platform to realize the algorithm.

References

[1] Lee, Sang Il, and Seong Joon Yoo. Threshold-based portfolio: the role of the threshold and its applications[J]. The Journal of Supercomputing, 2018: 1-18.

[2] Jia H. Investigation into the effectiveness of long short term memory networks for stock price prediction[J]. arXiv preprint arXiv:1603.07893, 2016.

[3] Christopher Olah. Understanding LSTM Networks.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/