使用LSTM預測股票收盤價
以PYTORCH實作

2019 6.27

107-2 PYTORCH與機器學習期末報告

Prepared by:

資管三　翁典

資管三　林貞妮

國貿碩二　劉鈞瑜

目錄

前言

影響股票市場波動的因素非常多，因此預測股價趨勢的不確定性很高，難以掌握其變化，身為高風險高報酬的金融商品有很多人將其當作投資標的或操作商品，故若能將深度學習技術中的類神經網路應用在股價預測中並取得一些進展對於廣大投資人應是一大福音。我們選取鈺德(3050)這檔中小型股票的收盤價作為資料的輸入並用一周 5 日交易日的收盤價預測下一天的收盤價。

Note：選取中小型股較不易受到大盤波動的影響

數據概述

資料來源：TWSE 台灣證券交易所 鈺德(3050)

抓取資料：從證交所網站上下載 3050 的股價資料，從民國 104 年 1 月到 108 年 6 月 11 日總共 1075 天的資料(1075 筆)

資料集中有多個變數——日期（date）、開盤價(open)、最高價(high)、最低價(low)、成交股數、收盤價(close)、成交筆數和漲跌價差、成交金額

- 開盤價和收盤價代表股票在某一天交易的起始價和最終價。
- 最高價、最低價和最後交易價表示當天股票的最高價、最低價和最後交易價格。

要注意的另一點是，市場在週末和公共假期休市，因此我們抓取共 1075 天的資料已扣除休市日。

A. 匯入資料集：將從證交所下載下來的每月股價資料 csv 檔使用 Excel 的新增查詢將數個月的資料合併成一個檔案，再匯入到 dataframe(df)。同時我們也將該股票(3050)每月的財務比率資料 csv 檔(包含殖利率、本益比、股價淨值比)下載並合併再匯入到 dataframe(df1)，後續會將兩者合併。

B. 將資料欄位(成交股數、成交金額、成交筆數)的資料型別從 object 轉換成 numeric 型態：

In [1]:
```
import torch
import pandas as pd
print(torch.__version__)
```

```
1.1.0
```

In [2]:
```
df=pd.read_csv(r'C:\Users\Danny Wong\Desktop\PyTorch\3050_data_new\3050_merge_New.csv',encoding='big5')
```

In [3]:
```
df1=pd.read_csv(r'C:\Users\Danny Wong\Desktop\PyTorch\3050_data_new\3050_PE_merge_New.csv',encoding='big5')
```

In [4]:
```
df
```

Out[4]:

	日期	成交股數	成交金額	開盤價	最高價	最低價	收盤價	漲跌價差	成交筆數
0	105/01/04	109,269	922,003	8.50	8.50	8.36	8.41	-0.11	72
1	105/01/05	371,004	3,137,752	8.38	8.63	8.31	8.51	0.1	101
2	105/01/06	363,001	3,138,448	8.60	8.74	8.51	8.51	0	108
3	105/01/07	550,003	4,627,854	8.45	8.57	8.20	8.33	-0.18	161
4	105/01/08	119,518	1,001,244	8.33	8.46	8.30	8.37	0.04	49
5	105/01/11	461,452	3,832,650	8.40	8.47	8.10	8.10	-0.27	145
6	105/01/12	520,064	4,220,795	8.16	8.28	7.90	7.90	-0.2	128
7	105/01/13	469,331	3,707,582	7.95	8.05	7.80	7.80	-0.1	149
8	105/01/14	279,900	2,159,295	7.79	7.82	7.60	7.70	-0.1	148
9	105/01/15	297,008	2,291,446	7.84	7.84	7.60	7.60	-0.1	76
10	105/01/18	186,025	1,428,459	7.52	7.74	7.52	7.74	0.14	48
11	105/01/19	156,002	1,222,474	7.77	7.90	7.77	7.80	0.06	64

In [6]:
```
i=0
j=0
k=0
for a in df['成交股數']:
    x=a.replace(",","")
    df['成交股數'][i]=x
    i=i+1

for b in df['成交金額']:
    y=b.replace(",","")
    df['成交金額'][j]=y
    j=j+1

for c in df['成交筆數']:
    if(isinstance(c, int)==False):
        z=c.replace(",","")
        df['成交筆數'][k]=z
    k=k+1
```

In [7]:
```
df['漲跌價差'] = df['漲跌價差'].apply(pd.to_numeric, errors='coerce')
df['成交股數'] = df['成交股數'].apply(pd.to_numeric, errors='coerce')
df['成交金額'] = df['成交金額'].apply(pd.to_numeric, errors='coerce')
df['成交筆數'] = df['成交筆數'].apply(pd.to_numeric, errors='coerce')
```

In [8]:
```
df.dtypes
```

Out[8]:
```
日期       object
成交股數      int64
成交金額      int64
開盤價     float64
最高價     float64
最低價     float64
收盤價     float64
漲跌價差    float64
成交筆數      int64
dtype: object
```

C. 將日期欄位的年份從民國年轉換至西元年：

In [9]:
```
c=0
for x in df['日期']:
    if(x[0:3]=='107'):
        y=x.replace("107","2018")
        df['日期'][c]=y
    elif(x[0:3]=='106'):
        z=x.replace("106","2017")
        df['日期'][c]=z
    elif(x[0:3]=='105'):
        q=x.replace("105","2016")
        df['日期'][c]=q
    elif(x[0:3]=='104'):
        r=x.replace("104","2015")
        df['日期'][c]=r
    else:
        s=x.replace("108","2019")
        df['日期'][c]=s
    c=c+1
```

```
In [10]: df
```

Out[10]:

	日期	成交股數	成交金額	開盤價	最高價	最低價	收盤價	漲跌價差	成交筆數
0	2016/01/04	109269	922003	8.50	8.50	8.36	8.41	-0.11	72
1	2016/01/05	371004	3137752	8.38	8.63	8.31	8.51	0.10	101
2	2016/01/06	363001	3138448	8.60	8.74	8.51	8.51	0.00	108
3	2016/01/07	550003	4627854	8.45	8.57	8.20	8.33	-0.18	161
4	2016/01/08	119518	1001244	8.33	8.46	8.30	8.37	0.04	49
5	2016/01/11	461452	3832650	8.40	8.47	8.10	8.10	-0.27	145
6	2016/01/12	520064	4220795	8.16	8.28	7.90	7.90	-0.20	128
7	2016/01/13	469331	3707582	7.95	8.05	7.80	7.80	-0.10	149
8	2016/01/14	279900	2159295	7.79	7.82	7.60	7.70	-0.10	148
9	2016/01/15	297008	2291446	7.84	7.84	7.60	7.60	-0.10	76
10	2016/01/18	186025	1428459	7.52	7.74	7.52	7.74	0.14	48
11	2016/01/19	156002	1222474	7.77	7.90	7.77	7.80	0.06	64

```
In [11]: c=0
for x in df1['日期2']:
    if(x[0:3]=='107'):
        y=x.replace("107","2018")
        df1['日期2'][c]=y
    elif(x[0:3]=='106'):
        z=x.replace("106","2017")
        df1['日期2'][c]=z
    elif(x[0:3]=='105'):
        q=x.replace("105","2016")
        df1['日期2'][c]=q
    elif(x[0:3]=='104'):
        p=x.replace("104","2015")
        df1['日期2'][c]=p
    else:
        r=x.replace("108","2019")
        df1['日期2'][c]=r
    c=c+1
```

D. 將日期欄位改成可以轉成 datetime 的形式：

```
In [12]: d=0
for x in df1['日期2']:
    y=x.replace('年','-')
    df1['日期2'][d]=y
    d=d+1
```

```
In [13]: d=0
for x in df1['日期2']:
    z=x.replace('月','-')
    df1['日期2'][d]=z
    d=d+1
```

```
In [14]: d=0
for x in df1['日期2']:
    u=x.replace('日','')
    df1['日期2'][d]=u
    d=d+1
```

E. 將日期資料型態轉換成 datetime 格式再將 df 與 df1 合併成新的 dataframe(result)：

```
In [15]: from datetime import datetime

df['日期']= pd.to_datetime(df['日期'], format='%Y/%m/%d')
df1['日期2']= pd.to_datetime(df1['日期2'], format='%Y/%m/%d')
#result['日期']
#result['收盤價']=result['收盤價'].astype("float32")
```

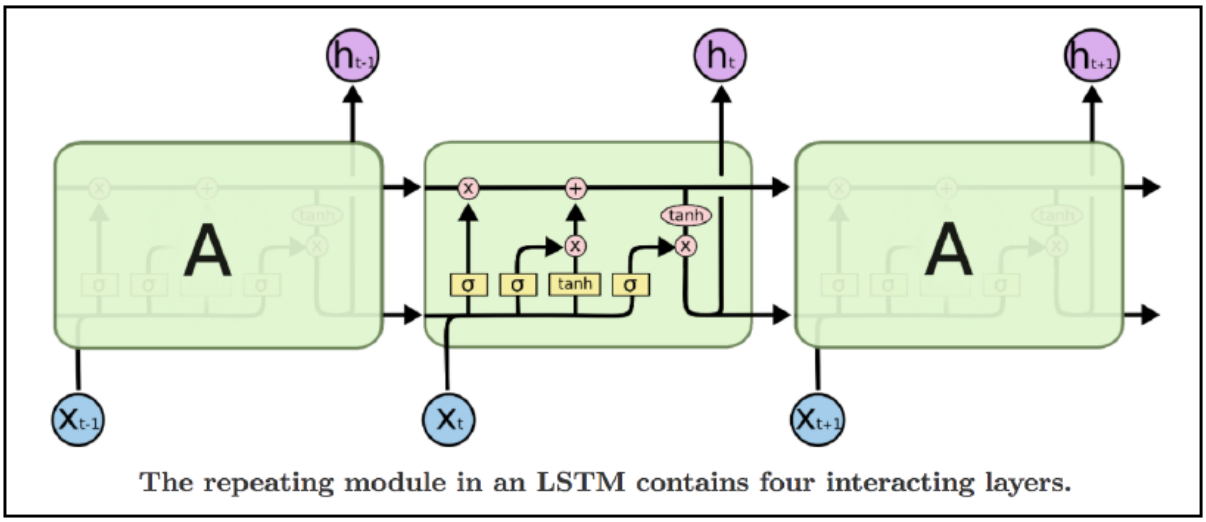
```
In [16]: df1.rename(columns={'日期2': '日期'}, inplace=True)
result = pd.merge(df, df1, on='日期')
```

```
In [17]: result
```

Out[17]:

	日期	成交股數	成交金額	開盤價	最高價	最低價	收盤價	漲跌價差	成交筆數	殖利率(%)	本益比	股價淨值比
0	2016-01-04	109269	922003	8.50	8.50	8.36	8.41	-0.11	72	38.23	0.0	0.57
1	2016-01-05	371004	3137752	8.38	8.63	8.31	8.51	0.10	101	38.68	0.0	0.58
2	2016-01-06	363001	3138448	8.60	8.74	8.51	8.51	0.00	108	38.68	0.0	0.58
3	2016-01-07	550003	4627854	8.45	8.57	8.20	8.33	-0.18	161	37.86	0.0	0.57
4	2016-01-08	119518	1001244	8.33	8.46	8.30	8.37	0.04	49	38.05	0.0	0.57
5	2016-01-11	461452	3832650	8.40	8.47	8.10	8.10	-0.27	145	36.82	0.0	0.55
6	2016-01-12	520064	4220795	8.16	8.28	7.90	7.90	-0.20	128	35.91	0.0	0.54
7	2016-01-13	469331	3707582	7.95	8.05	7.80	7.80	-0.10	149	35.45	0.0	0.53
8	2016-01-14	279900	2159295	7.79	7.82	7.60	7.70	-0.10	148	35	0.0	0.52
9	2016-01-15	297008	2291446	7.84	7.84	7.60	7.60	-0.10	76	34.55	0.0	0.52
10	2016-01-18	186025	1428459	7.52	7.74	7.52	7.74	0.14	48	35.18	0.0	0.53
11	2016-01-19	156002	1222474	7.77	7.90	7.77	7.80	0.06	64	35.45	0.0	0.53

模型分析



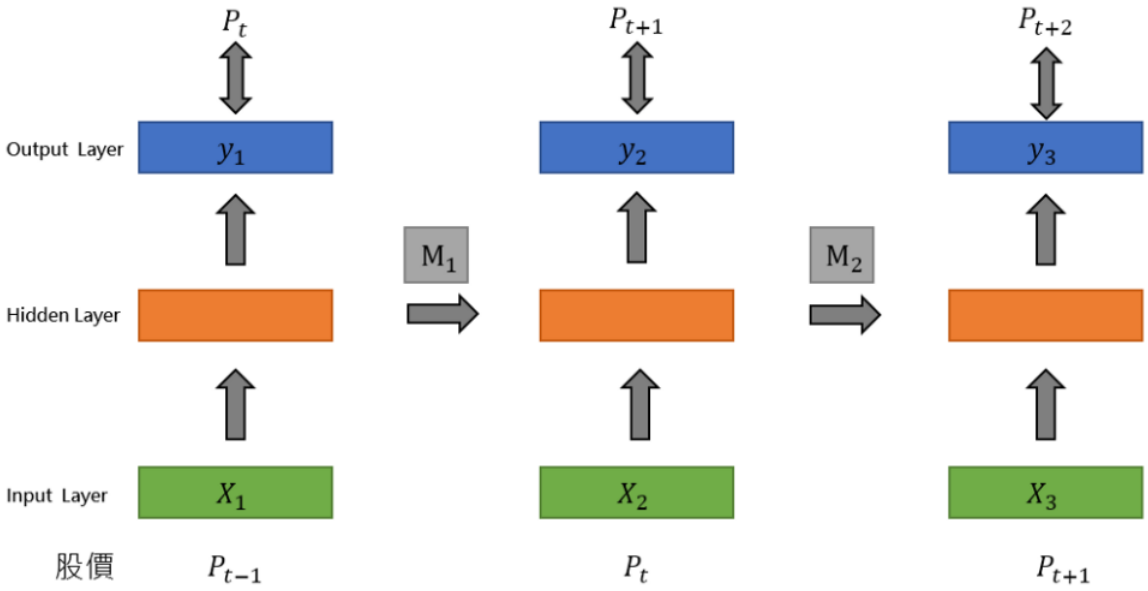
The repeating module in an LSTM contains four interacting layers.

金融類的數據基本都是時間序列數據(尤其是股票資料)，具有一定的先後順序。

LSTM，是目前 RNN(Recurrent Neural Network)中最常使用的模型。RNN 主要是要解決時間序列的問題。

一個簡單的 RNN 在學習股價上，我們可以將P_{t-1}的資料 input，並對P_t股價進行學習，而 M1 即是將 hidden output 存起來，並當我們要 train Pt，並針對P_{t+1}進行修正時，會將 M1 拿出來作考慮，並相加成為P_{t+1}的 output。

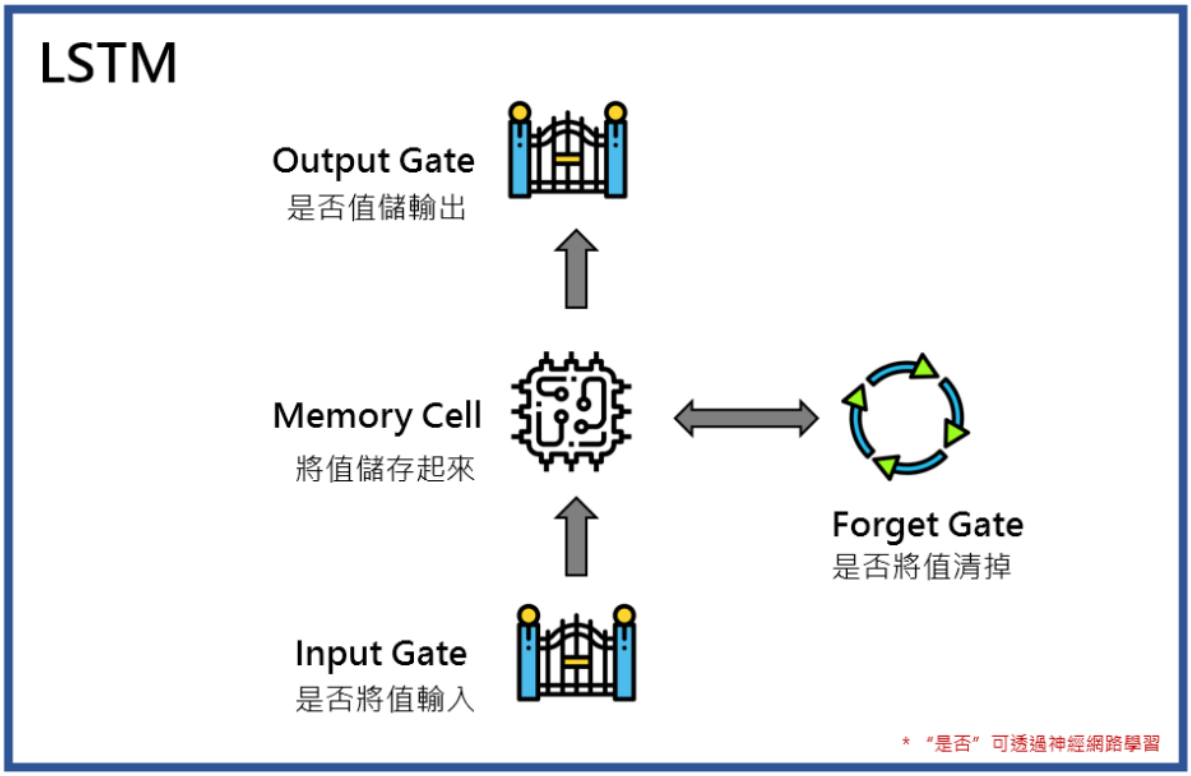
RNN



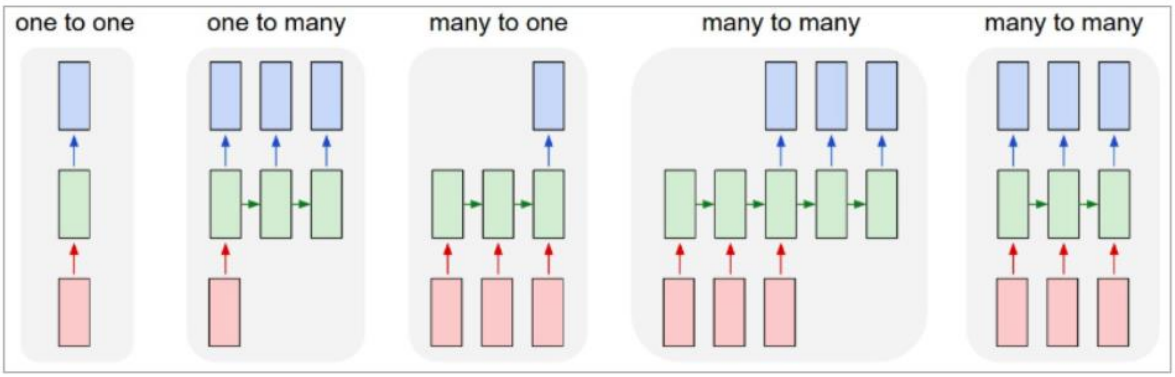
而 LSTM(Long short-term memory)，主要由四個 Component 組成: Input Gate、Output Gate、Memory Cell 以及 Forget Gate。

- Input Gate: 當將 feature 輸入時，input gate 會去控制是否將這次的值輸入
- Memory Cell: 將計算出的值儲存起來，以利下個階段拿出來使用
- Output Gate: 控制是否將這次計算出來的值 output
- Forget Gate: 是否將 Memory 清掉(format)，有點 restart 的概念。

其中，“是否”這件事情，是可以透過神經網路進行學習。



LSTM 有許多種變化，如下圖：



本實驗採 many to one 模式即 Input 為一周收盤價(sequence vector)

Output 為下一交易日的預測收盤價(Only one vector)

建構模型

●模型假設前提：我們**假設每天的收盤價依賴於當天的前 n 天之收盤價**
用 RNN 的 LSTM 模型來做為學習框架(對時間序列資料學習並預測股價)，讓模型學會用前 n 天的收盤價，來判斷當天的收盤價。**本文使用前一周(一周交易日)收盤價預測今天收盤價，所以 n=5**

A. 將最終合併資料按日期排序➜import 相關套件(包括 pandas、torch、numpy、matplotlib)

In [19]:
```
result=result.sort_values(by=['日期'])
```

In [20]:
```
import pandas as pd
import matplotlib.pyplot as plt
import datetime
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader
```

B. 撰寫兩個 function 分別為 generate_df_affect_by_n_days(產生時序資料 dataframe)及 readData(讀入 result 中的欄位資料並把日期作為 index 再將其切割為訓練資料與測試資料)

In [21]:
```
def generate_df_affect_by_n_days(series, n, index=False):
    if len(series) <= n:
        raise Exception("The Length of series is %d, while affect by (n=%d)." % (len(series), n))
    df = pd.DataFrame()
    for i in range(n):
        df['c%d' % i] = series.tolist()[i:-(n - i)]
    df['y'] = series.tolist()[n:]
    if index:
        df.index = series.index[n:]
    return df


def readData(column='收盤價', n=30, all_too=True, index=False, train_end=-300):
    df = result
    #df.index = list(df.index)

    df.index = list(df['日期'])
    df_column = df[column].copy()
    df_column_train, df_column_test = df_column[:train_end], df_column[train_end - n:]
    df_generate_from_df_column_train = generate_df_affect_by_n_days(df_column_train, n, index=index)
    if all_too:
        return df_generate_from_df_column_train, df_column, df.index.tolist()
    return df_generate_from_df_column_train
```

C. 建構神經網路模型類別並繼承 pytorch 內建的 RNN➜導入 LSTM 當作學習框架➜設定 LSTM 參數(input_size=1, hidden_size=90, num_layers=3)

參數說明：input_size=輸入矩陣特徵數，hidden_size=隱藏層的維度，num_layers=輸出層數

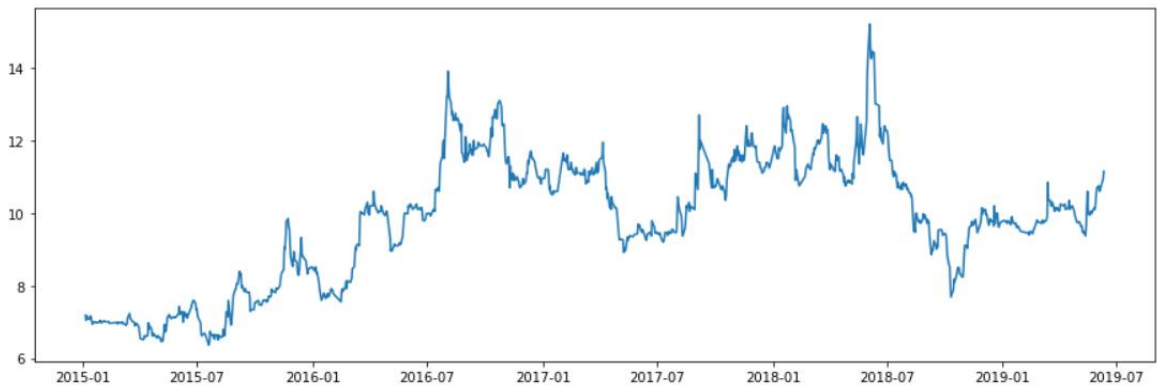
In [85]:
```
class RNN(nn.Module):
    def __init__(self, input_size):
        super(RNN, self).__init__()
        self.rnn = nn.LSTM(
            input_size=input_size,
            hidden_size=90,
            num_layers=3,
            batch_first=True
        )
        self.out = nn.Sequential(
            nn.Linear(90,3),
            nn.Linear(3,1)

        )

    def forward(self, x):
        r_out, (h_n, h_c) = self.rnn(x, None)  # None 表示 hidden state 会用全0的 state
        out = self.out(r_out)
        return out


class TrainSet(Dataset):
    def __init__(self, data):
        # 定义好 image 的路径
        self.data, self.label = data[:, :-1].float(), data[:, -1].float()

    def __getitem__(self, index):
        return self.data[index], self.label[index]

    def __len__(self):
        return len(self.data)
```

D. Hyperparameter(超參數)：本文以每周 5 日交易日收盤價預測下一天收盤價➜n=5, learning rate=LR=0.001, EPOCH=訓練批次量=100, 測試資料筆數=train_end=**-300**=總資料集**倒數 300 筆數**

In [107]:
```
n = 5
LR = 0.001
EPOCH = 100
train_end = -300
```

E. 建立數據集➜匯入資料到 numpy➜將資料標準化(μ =0, σ =1)➜將 numpy 放入 tensor 資料結構➜繪製原始資料圖

In [111]:
```
# 数据集建立
df, df_all, df_index = readData('收盤價', n=n, train_end=train_end)

df_all = np.array(df_all.tolist())
plt.figure(figsize=(15, 5))
plt.plot(df_index, df_all, label='real-data')

df_numpy = np.array(df)

df_numpy_mean = np.mean(df_numpy)
df_numpy_std = np.std(df_numpy)

df_numpy = (df_numpy - df_numpy_mean) / df_numpy_std
df_tensor = torch.Tensor(df_numpy)

trainset = TrainSet(df_tensor)
trainloader = DataLoader(trainset, batch_size=20, shuffle=True)
```



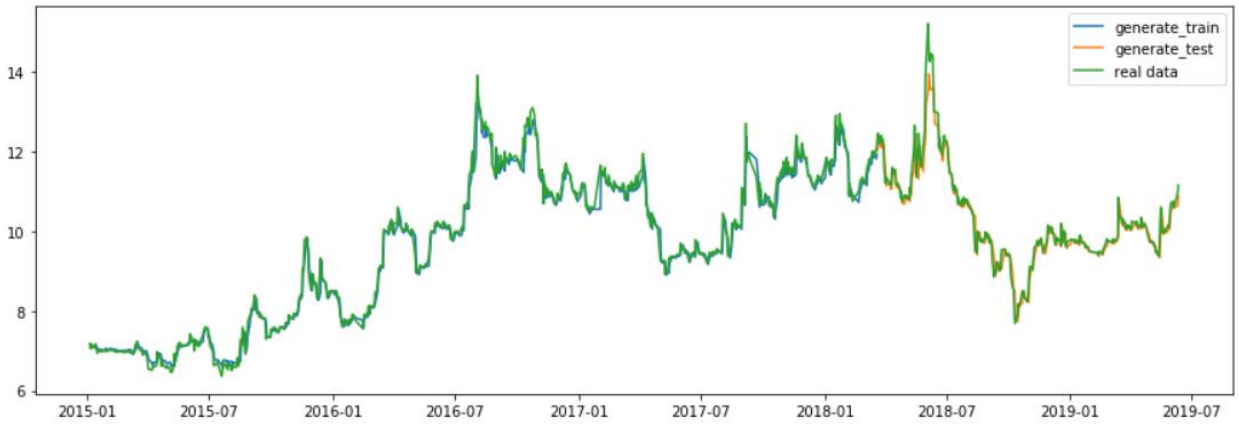
F. 訓練模型➜optimizer(學習優化器)使用 Adam(Adam 有做參數的偏離校正，使得每一次的學習率都會有個確定的範圍，會讓參數的更新較為平穩。)，loss 採用 MSE(均方誤差)並以 list L 儲存每次訓練 loss 值

In [112]:
```
rnn = RNN(n)
optimizer = torch.optim.Adam(rnn.parameters(), lr=LR)  # optimize all cnn parameters
loss_func = nn.MSELoss()
L=[]
for step in range(EPOCH):
    for tx, ty in trainloader:
        output = rnn(torch.unsqueeze(tx, dim=0))
        loss = loss_func(torch.squeeze(output), ty)
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # back propagation, compute gradients
        optimizer.step()
        L.append(loss)
    print(step, loss)
```

```
0 tensor(0.9089, grad_fn=<MseLossBackward>)
1 tensor(0.0355, grad_fn=<MseLossBackward>)
2 tensor(0.0351, grad_fn=<MseLossBackward>)
3 tensor(0.0390, grad_fn=<MseLossBackward>)
4 tensor(0.0074, grad_fn=<MseLossBackward>)
5 tensor(0.0143, grad_fn=<MseLossBackward>)
6 tensor(0.0453, grad_fn=<MseLossBackward>)
7 tensor(0.0221, grad_fn=<MseLossBackward>)
8 tensor(0.0237, grad_fn=<MseLossBackward>)
9 tensor(0.0124, grad_fn=<MseLossBackward>)
10 tensor(0.0366, grad_fn=<MseLossBackward>)
11 tensor(0.0066, grad_fn=<MseLossBackward>)
12 tensor(0.0166, grad_fn=<MseLossBackward>)
13 tensor(0.0148, grad_fn=<MseLossBackward>)
14 tensor(0.0088, grad_fn=<MseLossBackward>)
15 tensor(0.0104, grad_fn=<MseLossBackward>)
16 tensor(0.0197, grad_fn=<MseLossBackward>)
17 tensor(0.0127, grad_fn=<MseLossBackward>)
18 tensor(0.0099, grad_fn=<MseLossBackward>)
81 tensor(0.0220, grad_fn=<MseLossBackward>)
82 tensor(0.0306, grad_fn=<MseLossBackward>)
83 tensor(0.0041, grad_fn=<MseLossBackward>)
84 tensor(0.0244, grad_fn=<MseLossBackward>)
85 tensor(0.0396, grad_fn=<MseLossBackward>)
86 tensor(0.0095, grad_fn=<MseLossBackward>)
87 tensor(0.0047, grad_fn=<MseLossBackward>)
88 tensor(0.0040, grad_fn=<MseLossBackward>)
89 tensor(0.0557, grad_fn=<MseLossBackward>)
90 tensor(0.0185, grad_fn=<MseLossBackward>)
91 tensor(0.0069, grad_fn=<MseLossBackward>)
92 tensor(0.0108, grad_fn=<MseLossBackward>)
93 tensor(0.0044, grad_fn=<MseLossBackward>)
94 tensor(0.0151, grad_fn=<MseLossBackward>)
95 tensor(0.0138, grad_fn=<MseLossBackward>)
96 tensor(0.0055, grad_fn=<MseLossBackward>)
97 tensor(0.0172, grad_fn=<MseLossBackward>)
98 tensor(0.0034, grad_fn=<MseLossBackward>)
99 tensor(0.0070, grad_fn=<MseLossBackward>)
```

G. 繪製原始資料曲線與訓練資料及測試資料曲線(以評估預測成效)

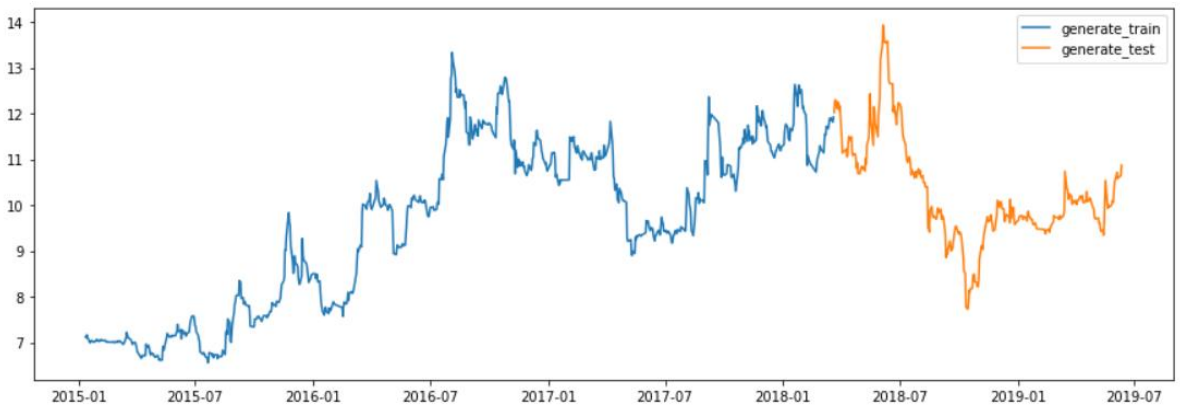
In [115]:
```
generate_data_train = []
generate_data_test = []

test_index = len(df_all) + train_end

df_all_normal = (df_all - df_numpy_mean) / df_numpy_std
df_all_normal_tensor = torch.Tensor(df_all_normal)
for i in range(n, len(df_all)):
    x = df_all_normal_tensor[i - n:i]
    x = torch.unsqueeze(torch.unsqueeze(x, dim=0), dim=0)
    y = rnn(x)
    if i < test_index:
        generate_data_train.append(torch.squeeze(y).detach().numpy() * df_numpy_std + df_numpy_mean)
    else:
        generate_data_test.append(torch.squeeze(y).detach().numpy() * df_numpy_std + df_numpy_mean)
plt.figure(figsize=(15, 5))
plt.plot(df_index[n:train_end], generate_data_train, label='generate_train')
plt.plot(df_index[train_end:], generate_data_test, label='generate_test')
plt.plot(result.index,result['收盤價'],label='real data')
plt.legend()
plt.show()
```



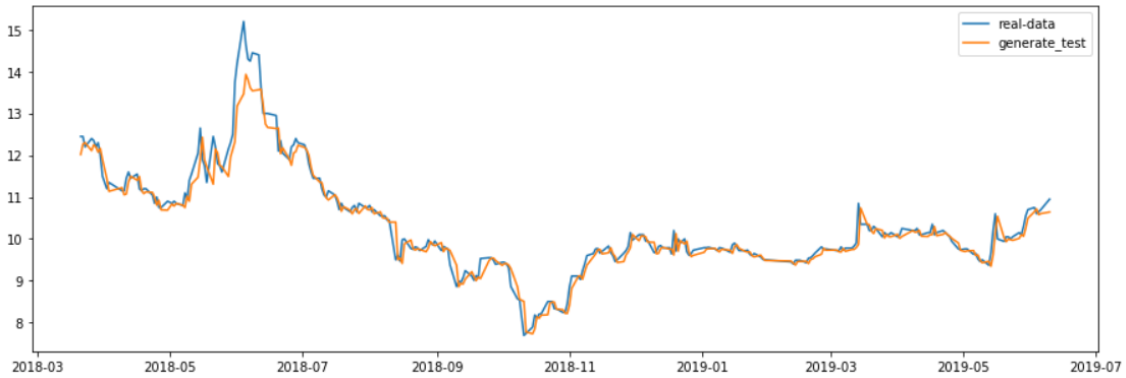
結論

A. 將訓練資料曲線以藍色呈現，測試資料以橘色呈現

In [116]:
```
plt.figure(figsize=(15, 5))
plt.plot(df_index[n:train_end], generate_data_train, label='generate_train')
plt.plot(df_index[train_end:], generate_data_test, label='generate_test')
plt.legend()
plt.show()
plt.figure(figsize=(15, 5))
plt.cla()
plt.plot(df_index[train_end:-1], df_all[train_end:-1], label='real-data')
plt.plot(df_index[train_end:-1], generate_data_test[:-1], label='generate_test')
plt.legend()
plt.show()
```



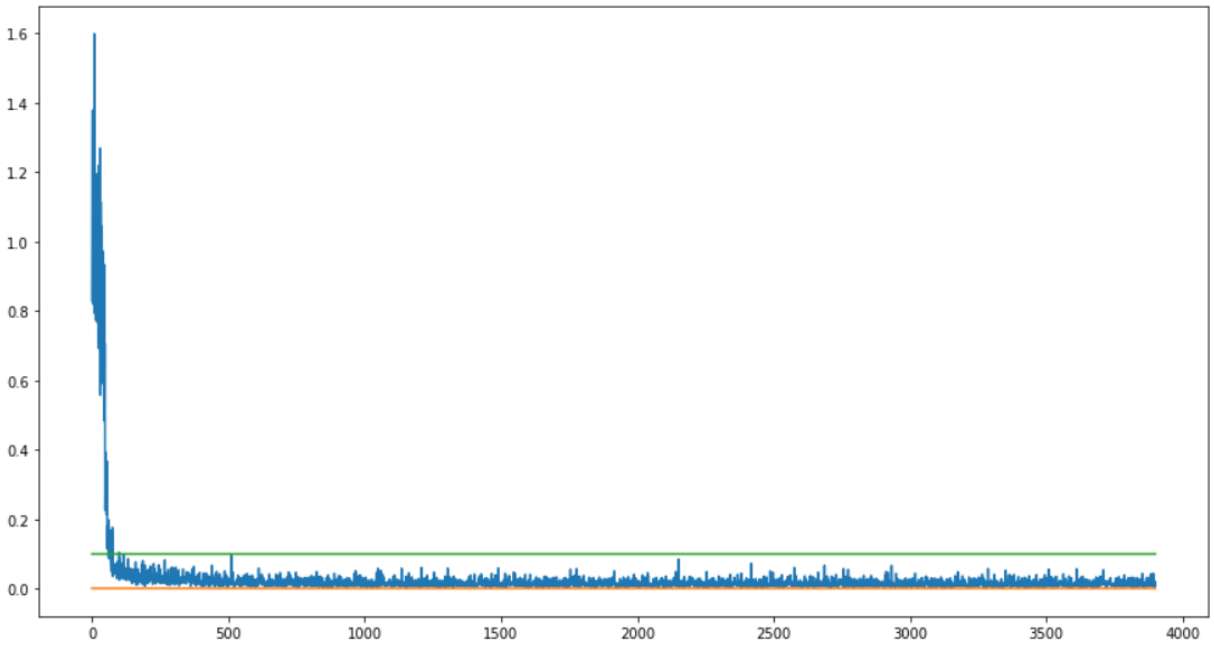
B. 繪製測試資料曲線與實際資料曲線比對訓練成效：從圖來看**趨勢大致正確但股價波動反應上稍有延遲**



C. 繪製 loss 值得圖形：從圖形來看 **loss 值從訓練起初大到訓練末期漸趨小還算有收斂**，**約莫在 loss=0.001~0.1 的區間擺盪**(upper bound=0.1 綠線所示, lower bound=0.001 橘線所示)

```
In [141]: z=[]
          U=[]
          for a in range(1,len(L)+1):
              z.append(0.001)
              U.append(0.1)
```

```
In [145]: plt.figure(figsize=(15, 8))
          plt.plot(L)
          plt.plot(z)
          plt.plot(U)
```



整體表現勉強可接受，未來也許透過**調整不同的批次量**、**增加特徵輸入(加入更多如其他價格或財務比率資訊)與訓練樣本數**、**優化模型參數**、**進行多次 cross validation** 或套**用不一樣的神經網路模型**能夠讓結果更加完善，經過此次嘗試發現仍有諸多進步空間，期望在未來能有所進步並讓深度學習在財金預測領域更加蓬勃發展。