Comparing LSTM, ARIMA, GARCH and CNN Model

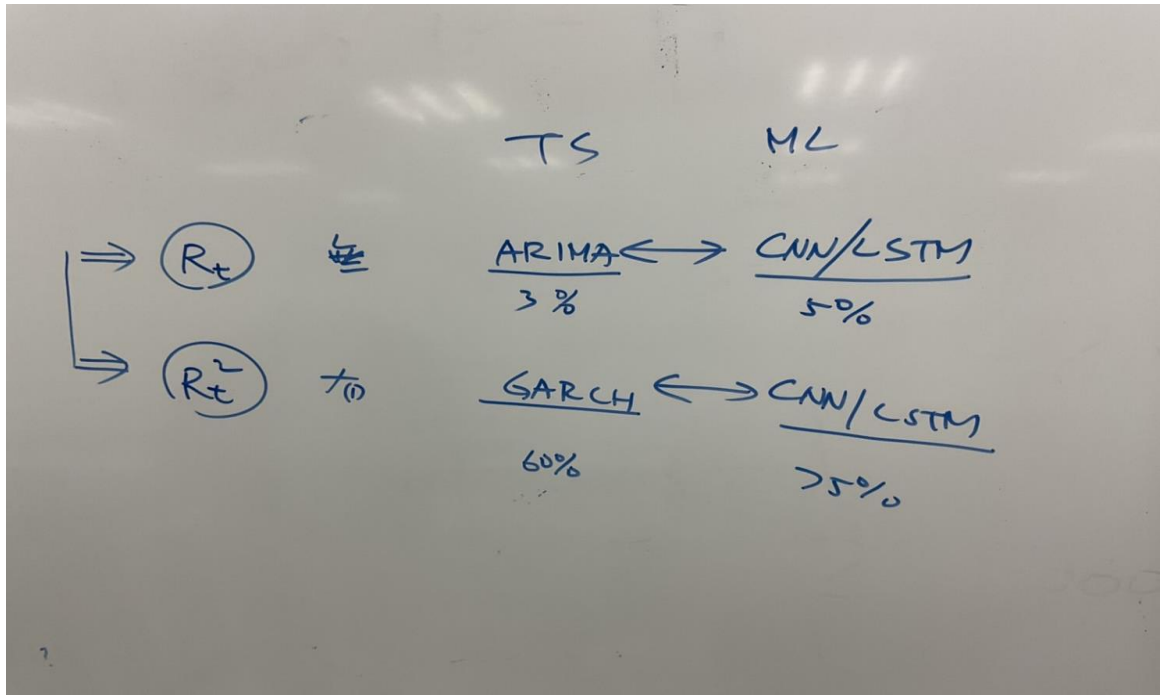
以台指期為例

Research Proposal

Github

呂偉丞

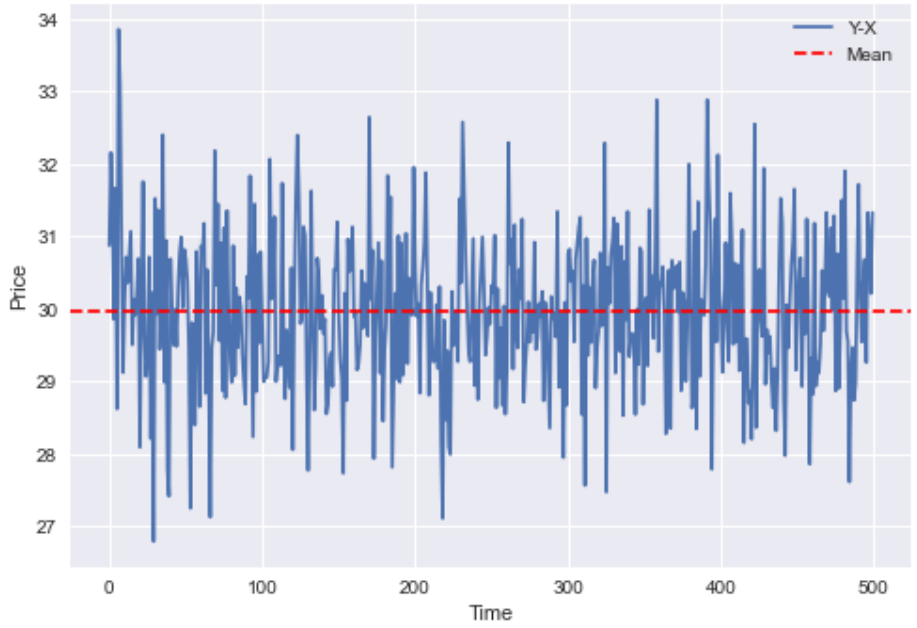
上次與老師討論重點



- 報酬率一階是uncorrelated，二階volatility 才具有關聯- volatility clustering
- 先用GARCH 與LSTM 分別分析volatility，再將兩者作結合

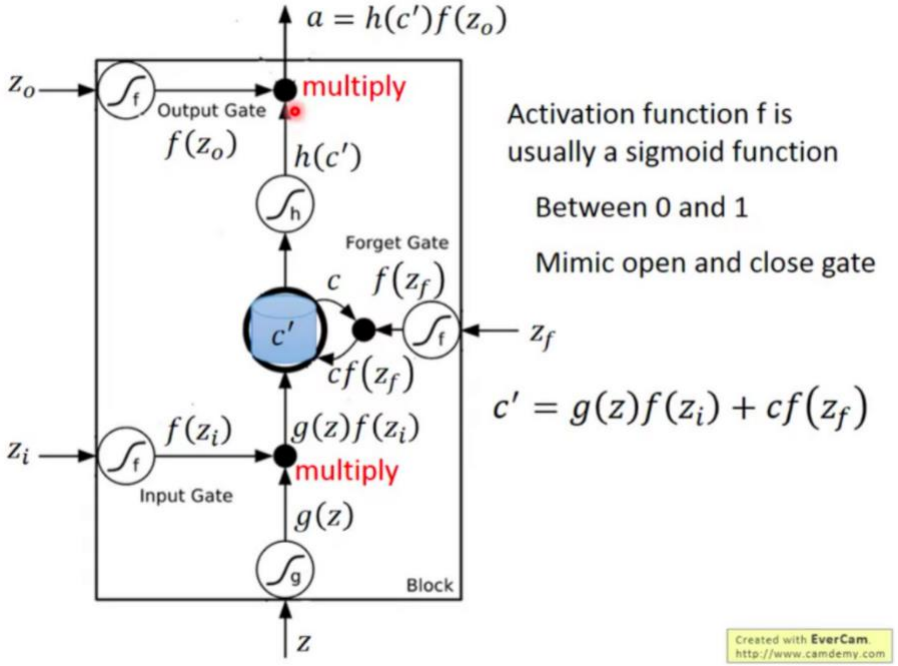
WHY LSTM

- LSTM 對於序列的預測十分強大，因為他能儲存過去的信息，包括時間序列
- LSTM 特別適合處理non-stationary 的時間序列像是股票
- LSTM 要做的事情就是找出一段時間區間的K棒當中有沒有重要訊號（如帶量紅K）並學習之後股價的走勢
- 找到數據在時間序列數據中的依賴關係

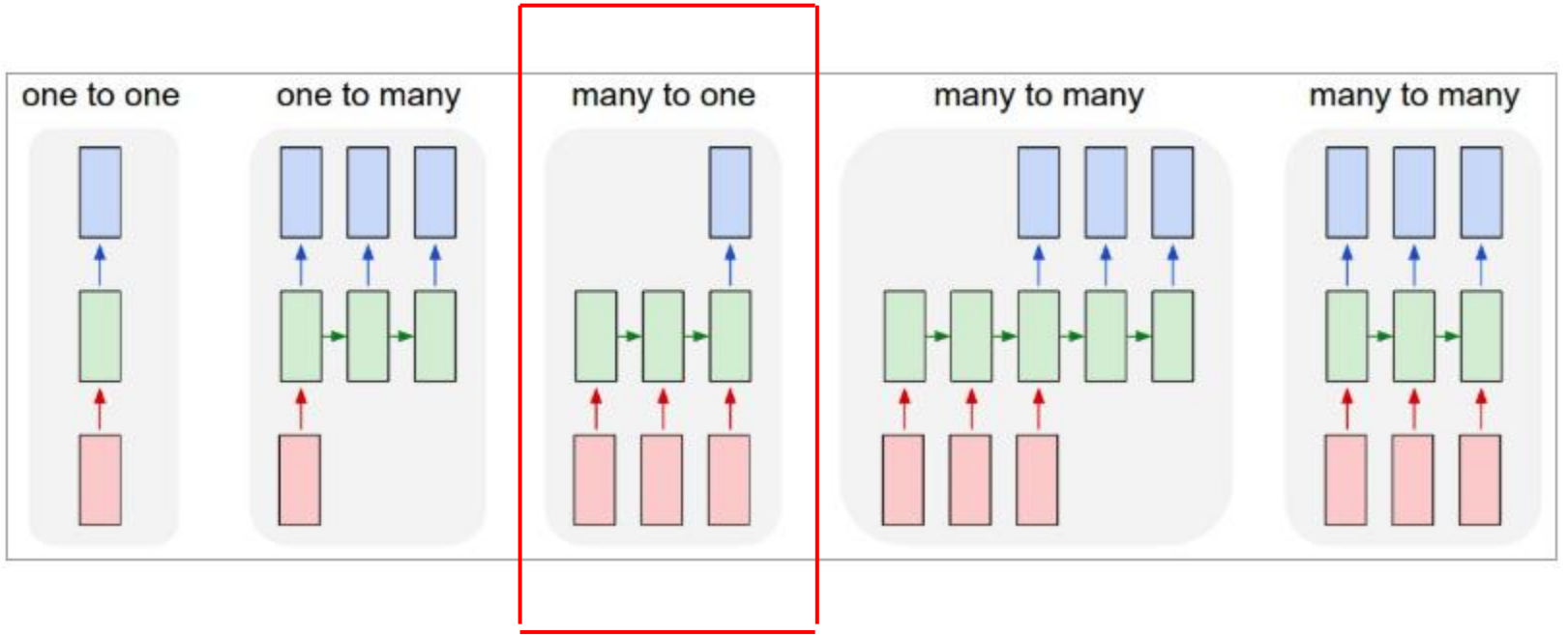


LSTM

- 主要由四個Component組成: Input Gate、Output Gate、Memory Cell以及Forget Gate。
  1. input Gate: 當將feature輸入時，input gate會去控制是否將這次的值輸入
  2. Memory Cell: 將計算出的值儲存起來，以利下個階段拿出來使用
  3. Output Gate: 控制是否將這次計算出來的值output
  4. Forget Gate: 是否將Memory清掉(format)



LSTM Model type



- 這次是使用 many to one的方式，用60天的收盤價預測下一天的股價

LSTM的資料維度

- Input data
- Time step
- Batch size

```
In [2]: df = web.DataReader('^TWII', data_source='yahoo', start='2015-01-01', end='2021-04-15')
        df
```

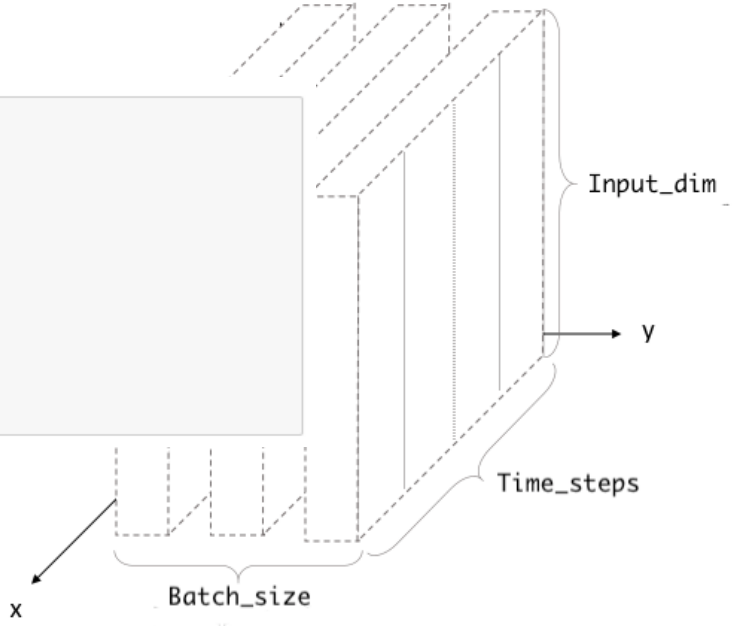
```
In [4]: sc = MinMaxScaler(feature_range = (0, 1))
        training_set_scaled = sc.fit_transform(training_set)

        X
        y    model.fit(X_train, y_train, epochs = 100, batch_size = 32)
        for i in range(60, 800):
            X_train.append(training_set_scaled[i-60:i, 0])
            y_train.append(training_set_scaled[i, 0])

        X_train, y_train = np.array(X_train), np.array(y_train)
        X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
```

```
2021-04-15  17076.730469  16851.060547  16851.060547  17076.730469  7006700.0  17076.730469
```

```
In [3]: training_set = df.iloc[:800, 5:6].values
        test_set = df.iloc[800:, 5:6].values
```



How to improve

Research Article
A CNN-LSTM-Based Model t

- 透過CNN提取特徵，再用LSTM做數據預測

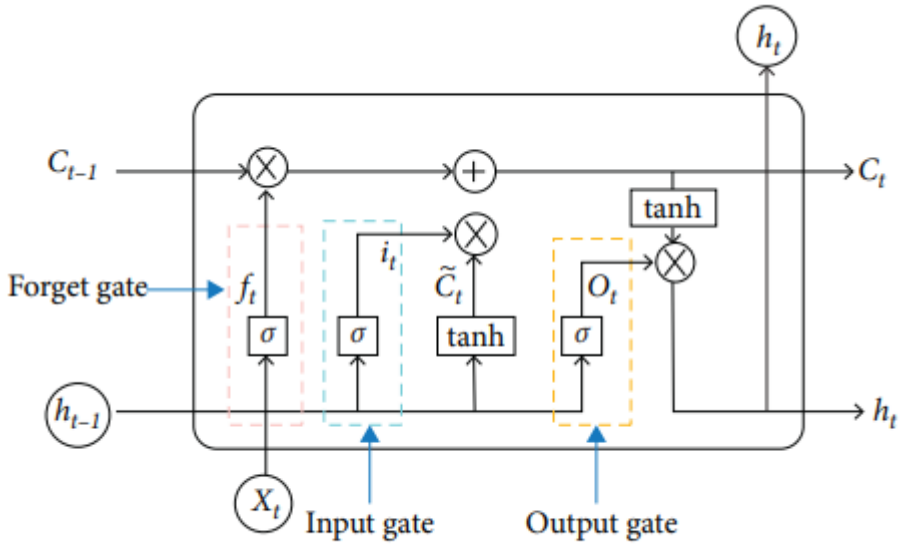
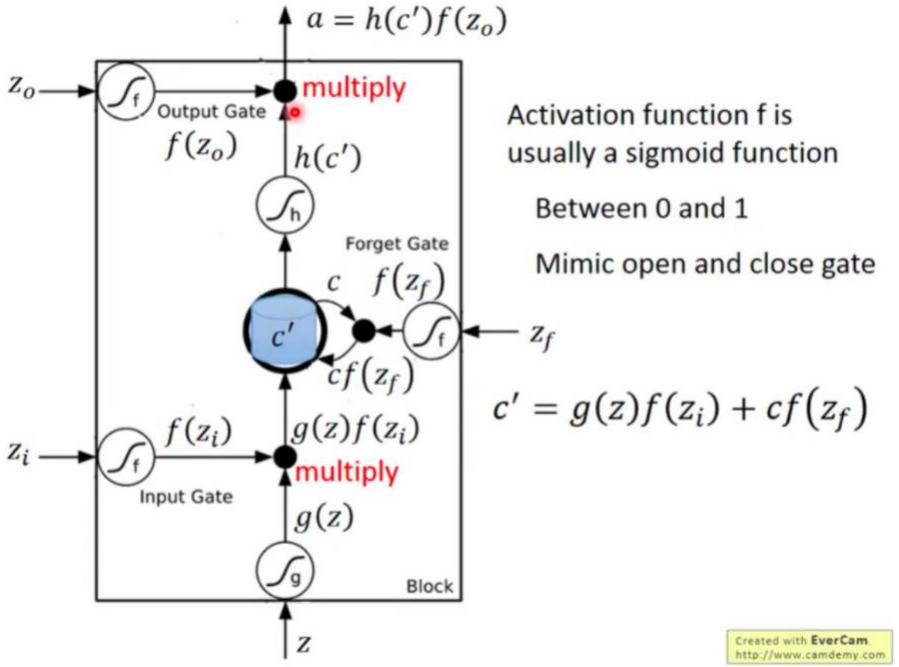


FIGURE 1: CNN-LSTM structure diagram.



How to improve

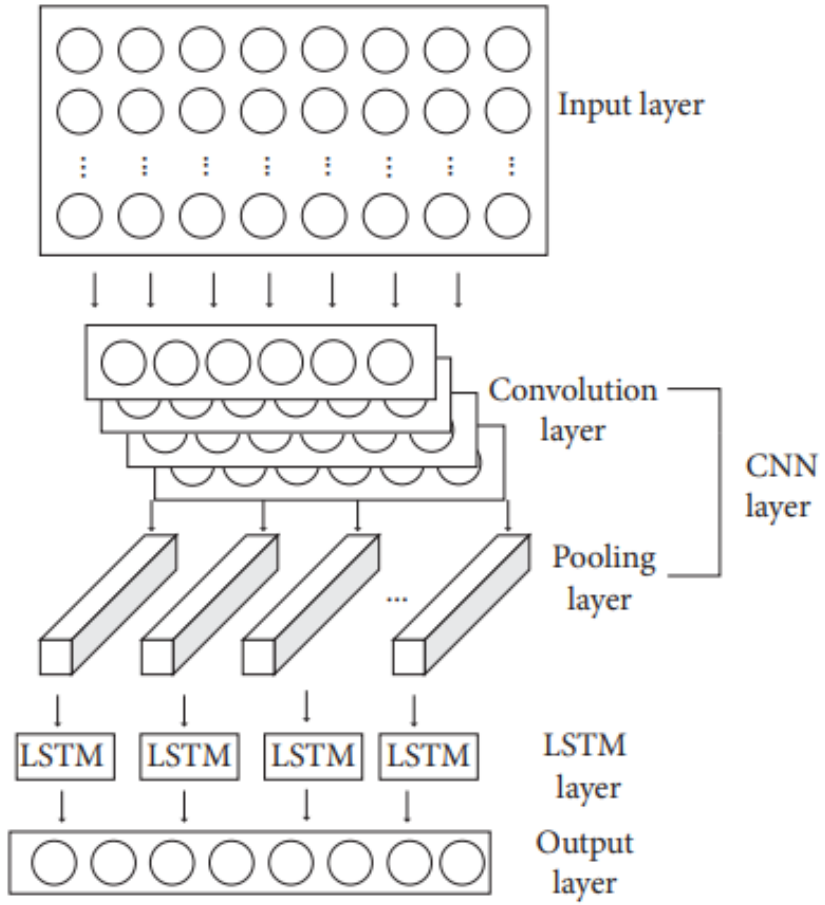
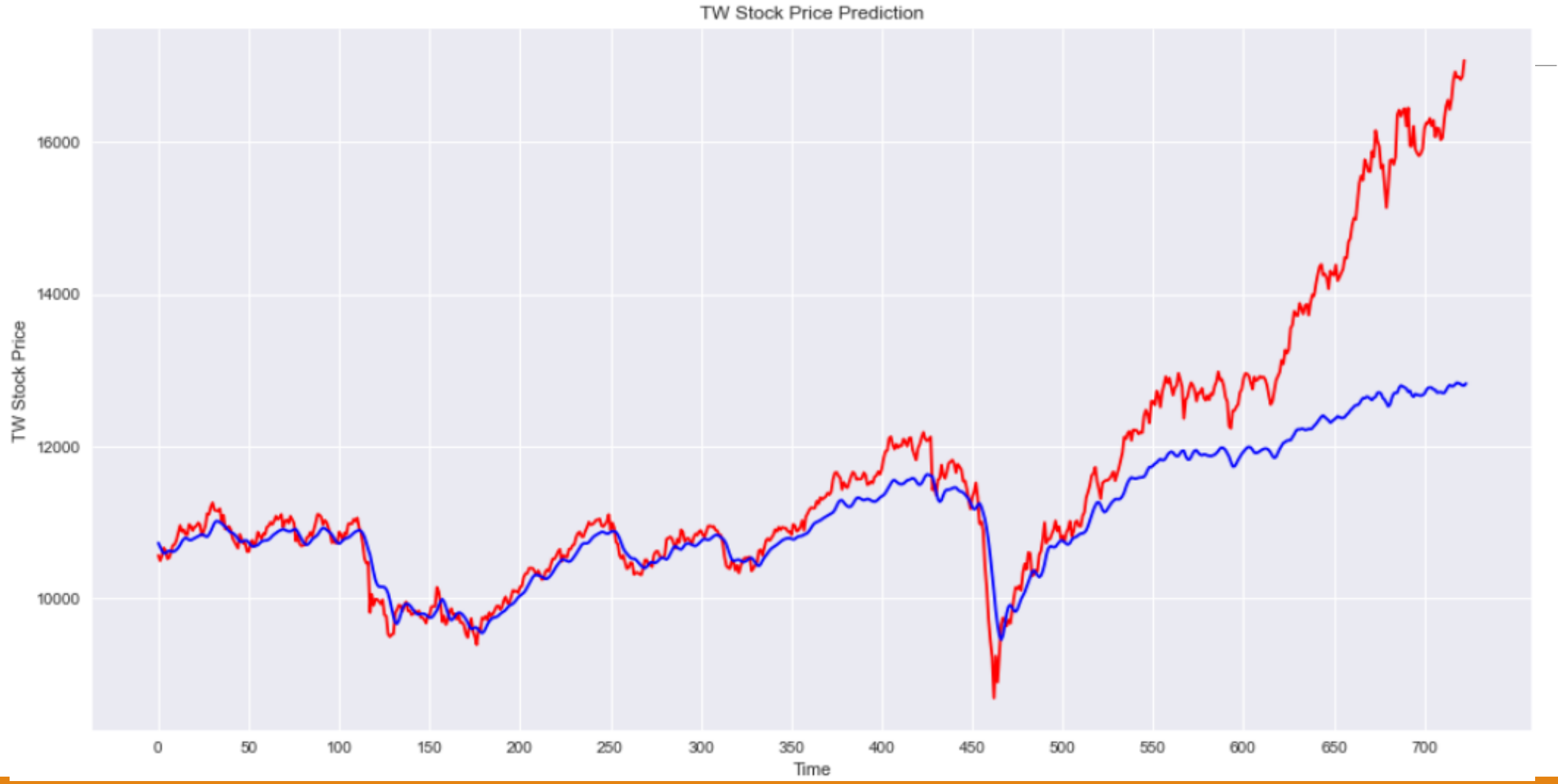


FIGURE 2: Architecture of LSTM memory cell.

Result


TW Stock Price Prediction
TW Stock Price
16000
14000
12000
10000
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
Time

前處理分析

1. 分析股票是否為平穩狀態，
若為長時間平穩，就難以在此作買進賣出。

2. 分析股票的離散程度

前處理分析-使用參數

以統計圖分析股票在時間上的性質:

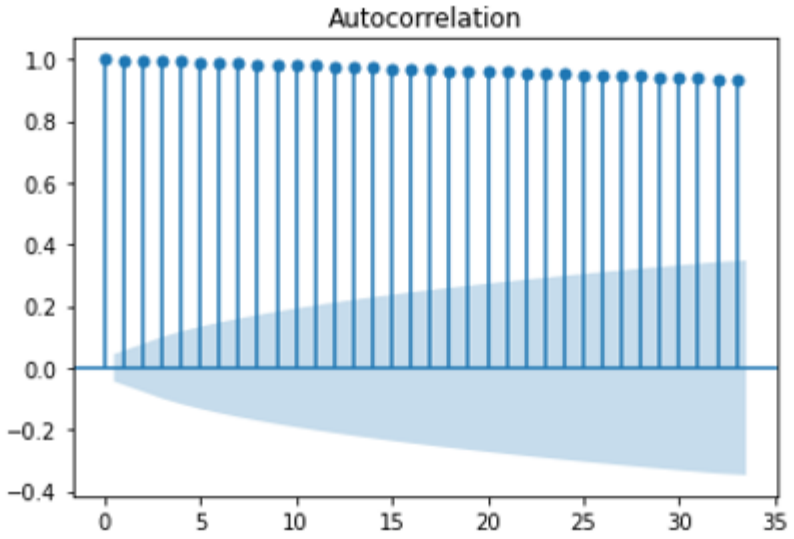
自我相關程度(**Autocorrelation**):

也叫**序列相關**，是一個訊號於其自身在不同時間點的**互相關**。非正式地來說，它就是兩次觀察之間的相似度對它們之間的時間差的函數。它是找出重複模式（如被噪聲掩蓋的週期訊號），或識別隱含在訊號諧波頻率中消失的基頻的數學工具。它常用於訊號處理中，用來分析函數或一系列值，如時時域訊號。

離散程度（**statistical dispersion**）:

在統計學裡，**離散程度**又稱**變異**或**變差**（英語：**variation**），是指一個分布或隨機變數的壓縮和拉伸程度。習慣上，離散程度更常用來描述分布，而變異更常用來描述隨機變數。用以描述離散程度或變異的量主要有變異數、標準差、變異係數和四分位距等。

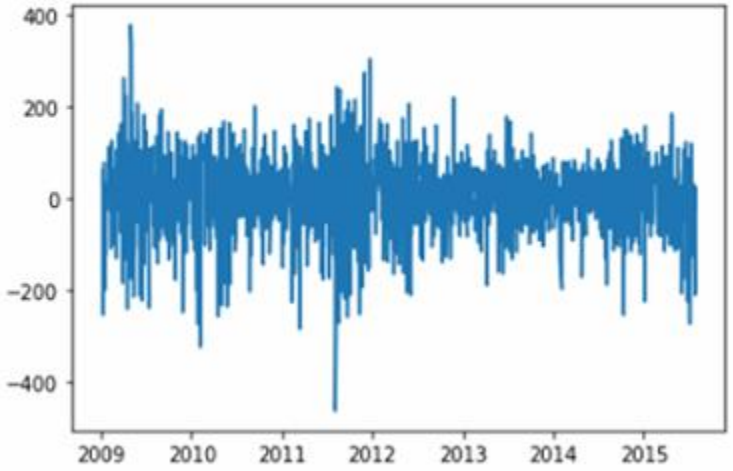
```
Out[7]: <function matplotlib.pyplot.show(close=None, block=None)>
```



根據自相關圖(Autocorrelation)，

該樣本序列具有一定趨勢性且初步判斷為非平穩時間序列。

```
In [4]: plt.plot(df_train['Close'].diff())
Out[4]: [<matplotlib.lines.Line2D at 0x20f75f260a0>]
```



接著透過.diff()來查詢給定軸上對象的離散差異，來辨別股市的浮動狀態。

時間序列模型: ARIMA模型

ARIMA 是一種單變量處理程序。資料序列的目前值與同一系列的過去值建立關聯以產生 AR 元件，又稱為 p。隨機誤差項的目前值與過去值建立關聯以產生 MA 元件 q。目前資料與過去資料的平均數和變異數假設是固定的，不隨時間變化。如果有需要，會增加 I 元件 (以 d 表示) 以透過差分修正不足的定態。

程式碼的參數架構 ARIMA (p、d、q)

具有下列3步驟:

$$\left(1-\sum_{i=1}^{p}\phi_i L^i\right)(1-L)^d X_t = \left(1+\sum_{i=1}^{q}\theta_i L^i\right)\varepsilon_t$$

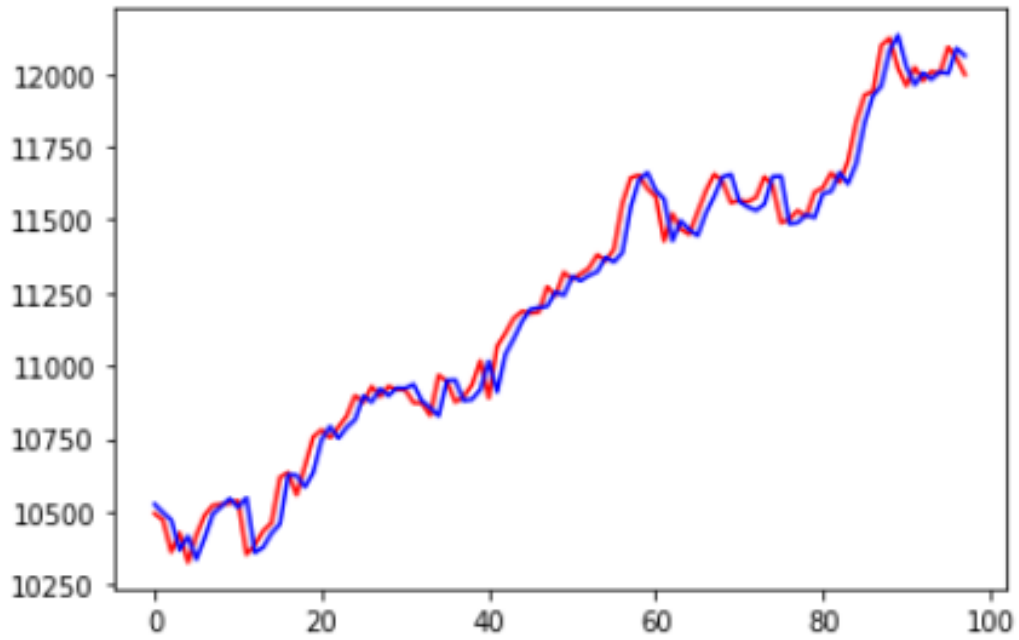
其中L 是滯後算子（Lag operator），$d \in \mathbb{Z}, d > 0$

1. 模型識別與選擇
2. 自我迴歸項 (AR)、整合或差分 (I)，以及移動平均項 (MA) 參數的預估
3. 模型檢查

Result

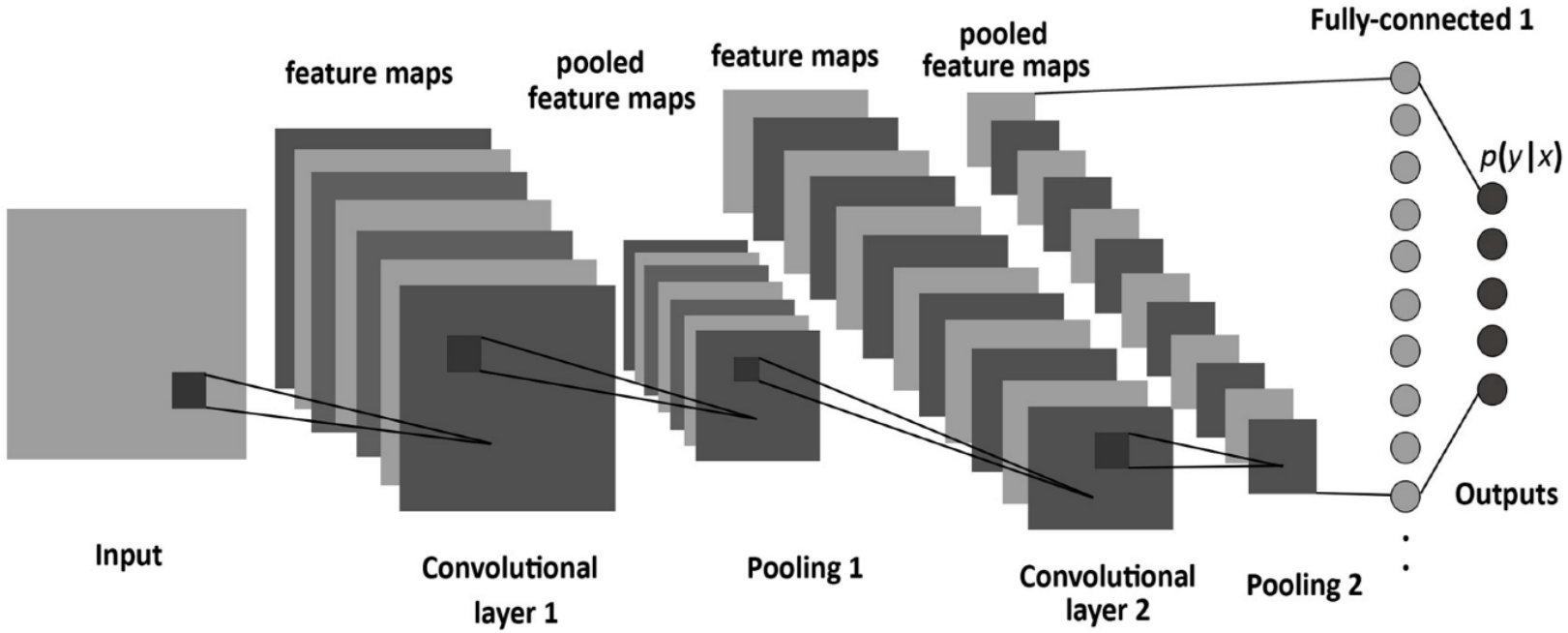
```
In [9]: plt.plot(test_close, color='red')
        plt.plot(preds, color='blue')
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x20f76145460>]
```



可以看到雖然幾乎是吻合，可是預測時間卻比實際狀況更晚發生，可能會導致該股票峰點已過卻預測錯誤。

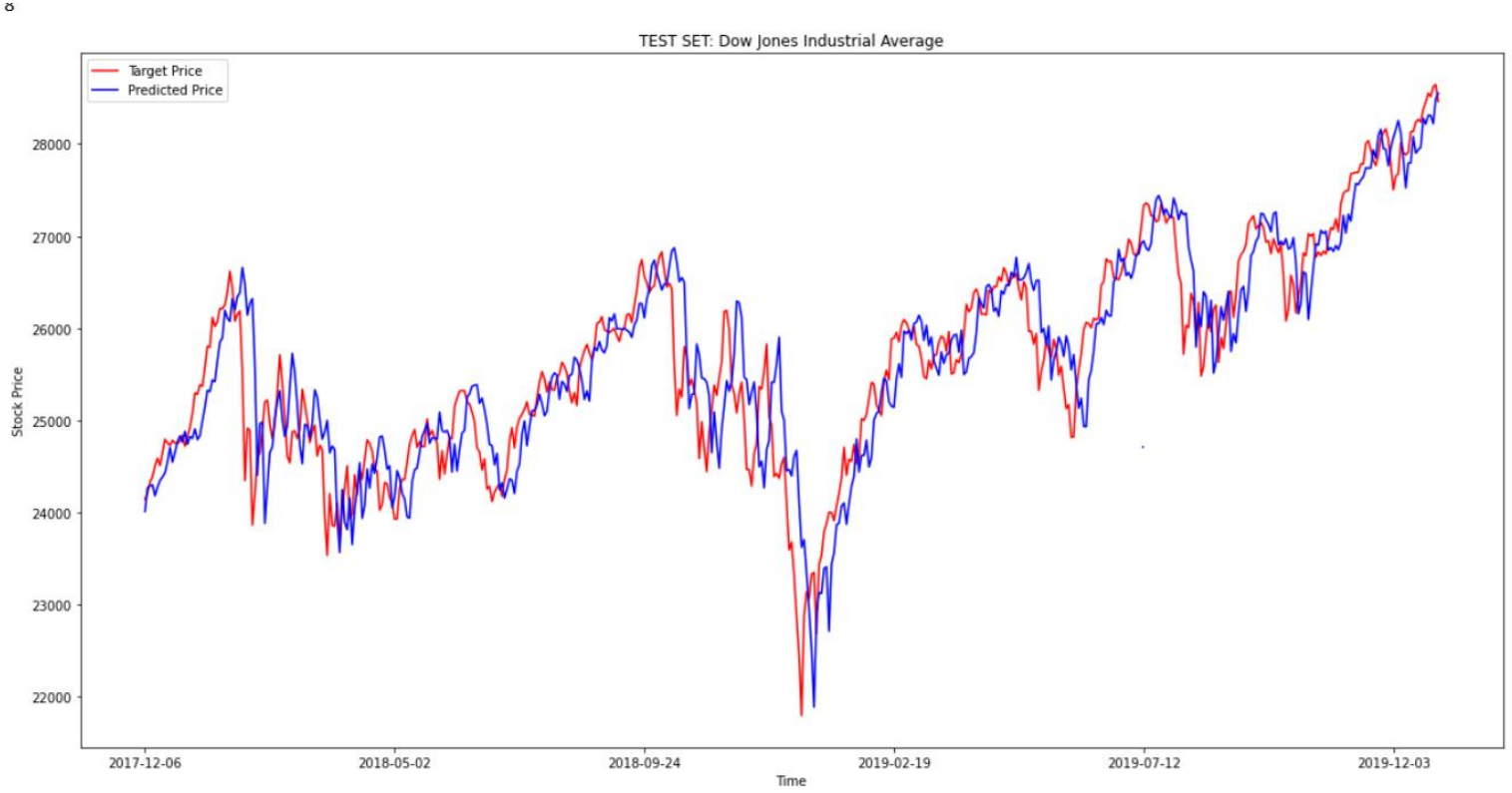
CNN模型



CNN多用於圖片，但若使用於股票的時間序列類型?

```
StockChartCNN(
  (conv): Conv2d(3, 32, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3))
  (batch_norm): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (max_pool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (relu): ReLU()
  (res_conv1): ResidualBlock(
    (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (batch_norm): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv3): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
    (relu): ReLU(inplace=True)
  )
  (res_conv2): ResidualBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (batch_norm): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv): Conv2d(128, 128, kernel_size=(1, 1), stride=(2, 2))
    (conv3): Conv2d(128, 256, kernel_size=(1, 1), stride=(1, 1))
    (relu): ReLU(inplace=True)
  )
  (res_conv3): ResidualBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1))
    (batch_norm): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (conv): Conv2d(256, 256, kernel_size=(1, 1), stride=(2, 2))
    (conv3): Conv2d(256, 512, kernel_size=(1, 1), stride=(1, 1))
    (relu): ReLU(inplace=True)
  )
  (average_pool): AvgPool2d(kernel_size=7, stride=7, padding=0)
  (layer_norm): LayerNorm((512, 1, 1), eps=1e-05, elementwise_affine=True)
  (fc1): Linear(in_features=512, out_features=500, bias=True)
  (dropout): Dropout(p=0.5, inplace=False)
  (fc2): Linear(in_features=500, out_features=100, bias=True)
  (fc3): Linear(in_features=100, out_features=25, bias=True)
  (out): Linear(in_features=25, out_features=1, bias=True)
)
```

Result



可以看出CNN在預測上甚至比ARIMA更不準確，而且偏移程度更嚴重，在股票預測上可能更失準，比不上時間序列的模型。